JBoss Enterprise Application Platform 5

Messaging User Guide

for use with JBoss Enterprise Application Platform 5

Tim Fox

Jeff Mesnil

Andy Taylor

Clebert Suconic

Howard Gao

JBoss Enterprise Application Platform 5 Messaging User Guide for use with JBoss Enterprise Application Platform 5 Edition 5.1.2

Author	Tim Fox
Author	Jeff Mesnil
Author	Andy Taylor
Author	Clebert Suconic
Author	Howard Gao
Editor	Laura Bailey
Editor	Jared Morgan

Copyright © 2011 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at *http://creativecommons.org/licenses/by-sa/3.0/*. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, MetaMatrix, Fedora, the Infinity Logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Java® is a registered trademark of Oracle and/or its affiliates.

XFS® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

All other trademarks are the property of their respective owners.

A guide to using JBoss Messaging 1.4 with the JBoss Enterprise Application Platform 5 and its patch releases.

Preface v
1. Document Conventions ... v
1.1. Typographic Conventions ... v
1.2. Pull-quote Conventions ... vi
1.3. Notes and Warnings ... vii
2. Getting Help and Giving Feedback ... vii
2.1. Do You Need Help? ... vii
2.2. Give us Feedback ... viii

1. About JBoss Messaging 1.4 1

2. Introduction 3
2.1. JBoss Messaging Features ... 3
2.2. Compatibility with JBoss MQ ... 4
2.3. System Properties used by JBoss Messaging ... 5
2.3.1. support.bytesId ... 5
2.3.2. retain.oldxabehaviour ... 5

3. JBoss Messaging Installation 7

4. Examples 9

5. Configuration 13
5.1. Configuring the ServerPeer ... 13
5.2. ServerPeer attributes ... 15
5.3. ServerPeer methods ... 17
5.4. Changing the Database ... 19
5.5. Configuring the Post Office ... 21
5.5.1. MessagingPostOffice Attributes ... 24
5.6. Configure the MessagingClusterHealthMBean ... 26
5.7. Configuring the Persistence Manager ... 27
5.7.1. JDBCPersistenceManager MBean Attributes ... 29
5.8. Configuring the JMS user manager ... 32
5.8.1. JMSUserManager MBean Attributes ... 33
5.9. Configuring Destinations ... 34
5.9.1. Pre-configured destinations ... 34
5.9.2. Configuring Queues ... 35
5.9.3. Configuring Topics ... 38
5.10. Configuring Connection Factories ... 42
5.10.1. ConnectionFactory Managed Bean Attributes ... 43
5.11. Configuring the Remoting Connector ... 46
5.12. ServiceBindingManager ... 49
5.13. Message Driven Beans ... 49

6. Clustering Notes 53
6.1. Unique Server Peer ID ... 53
6.2. Clustered Destinations ... 53
6.3. Clustered Durable Subscriptions ... 53
6.4. Clustered Temporary Destinations ... 53
6.5. Non-clustered Servers ... 53
6.6. Message Ordering in the Cluster ... 54
6.7. Idempotent Operations ... 54
6.8. Clustered Connection Factories ... 54

7. JBoss Messaging XA Recovery Configuration 55

8. JBoss Messaging Message Bridge Configuration 57
8.1. Message Bridge Overview ... 57

8.2. Bridge deployment 58
8.3. Bridge Configuration 58

9. Enabling JBoss Messaging Ordering Group **63**
9.1. Acknowledge Mechanism 63
9.2. How to Enable Message Ordering Group 63
9.2.1. Enabling Ordering Group on Producer 64
9.2.2. Enabling Ordering Group on Connection Factory 64
9.3. Notes and Limitations 65

A. Revision History **67**

Preface

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the *Liberation Fonts*[1] set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

1.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight keycaps and key combinations. For example:

> To see the contents of the file **`my_next_bestselling_novel`** in your current working directory, enter the **`cat my_next_bestselling_novel`** command at the shell prompt and press **`Enter`** to execute the command.

The above includes a file name, a shell command and a keycap, all presented in mono-spaced bold and all distinguishable thanks to context.

Key combinations can be distinguished from keycaps by the hyphen connecting each part of a key combination. For example:

> Press **`Enter`** to execute the command.
>
> Press **`Ctrl`**+**`Alt`**+**`F2`** to switch to the first virtual terminal. Press **`Ctrl`**+**`Alt`**+**`F1`** to return to your X-Windows session.

The first paragraph highlights the particular keycap to press. The second highlights two key combinations (each a set of three keycaps with each set pressed simultaneously).

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in **`mono-spaced bold`**. For example:

> File-related classes include **`filesystem`** for file systems, **`file`** for files, and **`dir`** for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialog box text; labeled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

> Choose **System** → **Preferences** → **Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click

[1] https://fedorahosted.org/liberation-fonts/

> **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).
>
> To insert a special character into a **gedit** file, choose **Applications** → **Accessories** → **Character Map** from the main menu bar. Next, choose **Search** → **Find…** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit** → **Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in proportional bold and all distinguishable by context.

Mono-spaced Bold Italic or ***Proportional Bold Italic***

Whether mono-spaced bold or proportional bold, the addition of italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

> To connect to a remote machine using ssh, type **ssh *username*@*domain.name*** at a shell prompt. If the remote machine is **example.com** and your username on that machine is john, type **ssh john@example.com**.
>
> The **mount -o remount *file-system*** command remounts the named file system. For example, to remount the **/home** file system, the command is **mount -o remount /home**.
>
> To see the version of a currently installed package, use the **rpm -q *package*** command. It will return a result as follows: ***package-version-release***.

Note the words in bold italics above — username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

> Publican is a *DocBook* publishing system.

1.2. Pull-quote Conventions

Terminal output and source code listings are set off visually from the surrounding text.

Output sent to a terminal is set in **mono-spaced roman** and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in **mono-spaced roman** but add syntax highlighting as follows:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;
```

```
public class ExClient
{
   public static void main(String args[])
       throws Exception
   {
      InitialContext iniCtx = new InitialContext();
      Object         ref    = iniCtx.lookup("EchoBean");
      EchoHome       home   = (EchoHome) ref;
      Echo           echo   = home.create();

      System.out.println("Created Echo");

      System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
   }
}
```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note

Notes are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring a box labeled 'Important' will not cause data loss but may cause irritation and frustration.

Warning

Warnings should not be ignored. Ignoring warnings will most likely cause data loss.

2. Getting Help and Giving Feedback

2.1. Do You Need Help?

If you experience difficulty with a procedure described in this documentation, visit the Red Hat Customer Portal at *http://access.redhat.com*. Through the customer portal, you can:

- search or browse through a knowledgebase of technical support articles about Red Hat products.
- submit a support case to Red Hat Global Support Services (GSS).

- access other product documentation.

Red Hat also hosts a large number of electronic mailing lists for discussion of Red Hat software and technology. You can find a list of publicly available mailing lists at *https://www.redhat.com/mailman/listinfo*. Click on the name of any mailing list to subscribe to that list or to access the list archives.

2.2. Give us Feedback

If you find a typographical error, or know how this guide can be improved, we would love to hear from you. Submit a report in Bugzilla against the product **JBoss Enterprise Application Platform 5** and the component **doc-JBoss_Messaging_User_Guide**. The following link will take you to a pre-filled bug report for this product: *http://bugzilla.redhat.com/*[2].

Fill out the following template in Bugzilla's **Description** field. Be as specific as possible when describing the issue; this will help ensure that we can fix it quickly.

```
Document URL:

Section Number and Name:

Describe the issue:

Suggestions for improvement:

Additional information:
```

Be sure to give us your name so that you can receive full credit for reporting the issue.

[2] https://bugzilla.redhat.com/enter_bug.cgi?product=JBoss%20Enterprise%20Application%20Platform%205&component=doc-JBoss_Messaging_User_Guide&version=5.1.2&short_desc=Bug%20in%20JBoss%20Messaging%20User%20Guide

About JBoss Messaging 1.4

JBoss Messaging is an enterprise messaging system from JBoss. It is a complete rewrite of JBossMQ, the legacy JBoss Java Message Service (JMS) provider.

JBoss Messaging is the default JMS provider in JBoss Enterprise Application Platform 4.3 and 5.

JBoss Messaging is integral to Red Hat's messaging strategy. It offers improvements to performance in both single node and clustered environments, and features a modular architecture so that we can easily add more features in the future.

This guide shows you how to install, set up, and configure JBoss Messaging for JBoss Enterprise Application Platform.

Introduction

JBoss Messaging provides an open-source and standards-based messaging platform to bring enterprise-class messaging to the mass market.

JBoss Messaging implements a robust, high-performance messaging core designed to support Service-Oriented Architectures (SOAs), Enterprise Service Buses (ESBs), and other integration requirements regardless of the level of demand.

JBoss Messaging lets you distribute application load evenly across your cluster. It balances each node's CPU cycles with no single point of failure, providing a highly scalable and performant clustering implementation.

JBoss Messaging includes a Java Messaging Service (JMS) front-end so that messages are delivered in a standards-based format, and to enable support for other messaging protocols in the future.

2.1. JBoss Messaging Features

JBoss Messaging provides the following features:

- A strong focus on performance, reliability and scalability with high throughput and low latency.
- A foundation for JBoss ESB for SOA initiatives. (JBoss ESB uses JBoss Messaging as its default JMS provider.)

JBoss Messaging also includes:

- publish-subscribe and point-to-point messaging models;
- persistent and non-persistent messages;
- guaranteed message delivery that ensures messages arrive once and only once where required;
- a transactional and reliable interface that supports ACID semantics;
- a customizable JAAS-based security framework;
- complete integration with JBoss Transactions (previously Arjuna JTA) to support full transaction recovery;
- an extensive JMX management interface;
- support for most major databases, including Oracle, DB2, Sybase, Microsoft SQL Server, PostgreSQL and MySQL;
- HTTP transport, for use with firewalls that allow only HTTP traffic;
- servlet transport to allow messaging through a dedicated servlet;
- SSL transport;
- configurable Dead Letter Queues (DLQs) and Expiry Queues;
- message statistics; which provide a rolling historical view of the messages delivered to queues and subscriptions;
- the automatic paging of messages to storage, which lets you use very large queues that would be too large to fit entirely within system memory; and

- strict message ordering which results in messages belonging to a particular message group being delivered according to the order of their arrival at the target queue.

JBoss Messaging also includes the following clustering features:

- **Fully-clustered queues and topics**
  Logical queues and topics are distributed across the cluster. You can send or receive a queue or topic to or from any node on the cluster.
- **Fully-clustered durable subscriptions**
  A particular durable subscription can be accessed from any node of the cluster, letting you spread processing load from that subscription across the entire cluster.
- **Fully-clustered temporary queues**
  If a sent message includes the **`replyTo`** of a temporary queue, it can be returned on any node of the cluster.
- **Intelligent message redistribution**
  Messages are automatically moved between nodes of the cluster to take advantage of different consumer speeds on different nodes. This helps to prevent starvation or build-up of messages on a particular node.
- **Message order protection**
  Enable this to ensure that the order of messages produced by a producer is identical to the order of messages consumed by a consumer. This works even if message redistribution is active.
- **Completely transparent failover**
  When a server fails, your sessions continue exception-free on a new node. This is also completely configurable: if you do not want to implement this failover behavior, you can disable it and fall back to exceptions being thrown and manually recreating connections on a new node.
- **High availability and seamless failover**
  If the node fails, you will automatically failover to a different node without losing any persistent messages and can seamlessly continue your session. *Once and only once* delivery of persistent messages is respected at all times.
- **Message bridge**
  JBoss Messaging contains a message bridge component, which lets you bridge messages between any two JMS 1.1 destinations. This lets you connect geographically separate clusters and form large, globally-distributed logical queues and topics.

2.2. Compatibility with JBoss MQ

JBoss MQ was the JMS implementation shipped with Enterprise Application Platform 4.2. Since JBoss Messaging is compatible with both JMS 1.1 and JMS 1.0.2b, the JMS code written against JBoss MQ will run with JBoss Messaging without any further changes.

JBoss Messaging has no wire format compatibility with JBoss MQ. It is therefore necessary to upgrade JBoss MQ clients with JBoss Messaging client JARs.

Important

Although JBoss Messaging deployment descriptors are similar to JBoss MQ deployment descriptors, they are *not identical*, and will require some simple adjustments before they will work with JBoss Messaging. The database data model is completely different, so JBoss Messaging should not be used with a JBoss MQ data schema, or vice-versa.

2.3. System Properties used by JBoss Messaging

2.3.1. support.bytesId

This system property controls the default behavior when constructing a **`JBossMessage`** object from a foreign message object. Set this property via the command line on server startup with the **`-D`** option.

If this property is set to **`true`**, the **`JBossMessage`** constructor will try to extract the native byte[] correlation ID from the foreign message headers. If set to **`false`**, it will use the normal string type **`JMSCorrelationID`**. This property will default to **`true`** if not set or when set to something other than **`true`** or **`false`**.

2.3.2. retain.oldxabehaviour

This system property controls the type of exception thrown by a JMS XAResource in the event that the `prepare()` method is called after the connection is broken. Set this property via the command line on server startup with the **`-D`** option.

If this property is not defined, an **`XAException`** with an **`XA_RBCOMMFAIL`** error code will be thrown. Otherwise an **`XAException`** with an **`XA_RETRY`** error code will be thrown. It should be noted that JBoss Messaging does not define this property by default.

JBoss Messaging Installation

JBoss Enterprise Application Platform (EAP) comes with JBoss Messaging pre-installed as the default JMS provider. If you are using EAP version 4.3 or higher, there is no need to manually install JBoss Messaging.

Examples

JBoss Messaging has a number of examples that are available for download. Download the examples archive file from *https://access.redhat.com*

Task: Download JBoss Messaging Examples Zip

Follow this task to download the JBoss Messaging Example Zip bundle. The examples are contained within the documentation bundle for the platform.

Prerequisites

- You have the correct entitlements for JBoss Enterprise Application Platform on access.redhat.com

1. Log in to the *Red Hat Customer Portal*[1].
2. Select **Downloads** → **JBoss Enterprise Middleware** → **Downloads**
3. On the Software Downloads page, select **Application Platform** from the **Product** drop-down menu.

   The Version drop-down menu defaults to the latest release.
4. Locate the **Application Platform *[version]* Documentation** entry, and click **Download**.
5. The documentation bundle begins to download.

Related Information

- *Task: Unpack and Deploy Examples*

Task: Unpack and Deploy Examples

Complete this task to unpack the JBoss Messaging examples from the platform documentation bundle, and meet all basic configuration requirements to run the examples.

- *Task: Download JBoss Messaging Examples Zip*
- A running JBoss Enterprise Application Server instance with default settings.

1. Open the zip archive using an archive utility appropriate for your operating system.
2. In the zip archive manager, navigate to **jboss-eap-5.1** → **doc**.
3. Extract the examples directory to **`$JBOSS_HOME/docs/examples`**.
4. Open **`$JBOSS_HOME/docs/examples/jboss-messaging-examples/destinations/`** in a file browser.
5. Copy **`jbm-examples-destinations-service.xml`** to **`$JBOSS_HOME/server/default/deploy`** to deploy the destinations configuration directives required by the examples.

Related Information

- *Unclustered Examples*
- *Clustered Examples*

Unclustered Examples

Important

You must run the Unclustered examples on JBoss Enterprise Application Platform non-clustered profiles: the **`All`** and **`Production`** profiles are not supported.

The **`readme.html`** for each example provides the setup details, expected output, and simple troubleshooting.

queue
: This example shows a simple send and receive to a remote queue using a JMS client

topic
: This example shows a simple send and receive to a remote topic using a JMS client

mdb
: This example demonstrates usage of an EJB2.1 MDB with JBoss Messaging

ejb3mdb
: This example demonstrates usage of an EJB3 MDB with JBoss Messaging

stateless
: This example demonstrates an EJB2.1 stateless session bean interacting with JBoss Messaging

mdb-failure
: This example demonstrates rollback and redelivery occuring with an EJB2.1 MDB

secure-socket
: This example demonstrates a JMS client interacting with a JBoss Messaging server using SSL encrypted transport

http
: This example demonstrates a JMS client interacting with a JBoss Messaging server tunneling traffic over the HTTP protocol

web-service
: This example demonstrates JBoss web-service interacting with JBoss Messaging

stateless-clustered
: This example demonstrates a JMS client interacting with clustered EJB2.1 stateless session bean, which in turn interacts with JBoss Messaging. The example uses HAJNDI to lookup the connection factory

bridge
: This example demonstrates using a message bridge. It deploys a message bridge in JBoss AS which then proceeds to move messages from a source to a target queue

servlet
: This example demonstrates how to use servlet transport with JBoss Messaging. It deploys a servlet and a ConnectionFactory that uses the servlet transport.

ordering-group
: This example demonstrates using strict message ordering with JBoss Messaging. It uses JBoss Messaging ordering group API to deliver strictly ordered messages, regardless of their priorities.

Clustered Examples

Important

The clustered examples require two running JBoss Application Server instances with port settings set to ports-01 and ports-02.

The examples are supported for use on the Enterprise Application Platform **`All`** and **`Production`** server profiles.

The **`readme.html`** for each example provides the setup details, expected output, and simple troubleshooting.

distributed-topic
: This example demonstrates a JMS client interacting with a JBoss Messaging distributed topic - it requires two JBoss AS instances to be running

distributed-queue
: This example demonstrates a JMS client interacting with a JBoss Messaging distributed queue - it requires two JBoss AS instances to be running

queue-failover
: This example demonstrates the transparent failover of a JMS consumer.

Configuration

The Java Message Service (JMS) API specifies how a messaging client interacts with a messaging server. How messaging services such as message destinations and connection factories are defined and implemented depends on the JMS provider. JBoss Messaging has its own files for service configuration.

This chapter shows you how to configure various services available in JBoss Messaging that work together to provide JMS API-level services to client applications.

JBoss Messaging configuration is divided between several configuration files. Depending on the type of service provided, configuration information is divided between **`messaging-service.xml`**, **`remoting-bisocket-service.xml`**, **`<your database type>-persistence-service.xml`**, **`connection-factories-service.xml`** and **`destinations-service.xml`**. These files can all be found in the **`$JBOSS_HOME/server/$PROFILE/deploy/messaging`** directory.

AOP interceptor stacks can be configured in **`aop-messaging-client.xml`** (for client-side behavior) and **`aop-messaging-server.xml`** (for server-side behavior). There is usually no need to change these files, but some interceptors can be removed to improve performance if they are not required. Ensure that you have considered the security implications before removing the security interceptor.

5.1. Configuring the ServerPeer

The **`ServerPeer`** is the heart of the JBoss Messaging JMS facade. You can configure its behavior by altering **`$JBOSS_HOME/server/$PROFILE/deploy/messaging/messaging-service.xml`**.

All JBoss Messaging services are based in the **`ServerPeer`**.

An example of a Server Peer configuration is presented below. Note that not all values for the server peer's attributes are specified in the example

```
<!-- ServerPeer MBean configuration
    ============================== -->

<mbean code="org.jboss.jms.server.ServerPeer"
  name="jboss.messaging:service=ServerPeer"
  xmbean-dd="xmdesc/ServerPeer-xmbean.xml">

  <!--The unique id of the server peer - in a cluster each node
      MUST have a unique value - must be an integer-->

  <attribute name="ServerPeerID">
    ${jboss.messaging.ServerPeerID:0}
  </attribute>

  <!--The default JNDI context to use for queues when they are
      deployed without specifying one-->

  <attribute name="DefaultQueueJNDIContext">/queue</attribute>

  <!--The default JNDI context to use for topics when they are
      deployed without specifying one -->

  <attribute name="DefaultTopicJNDIContext">/topic</attribute>

  <attribute name="PostOffice">
    jboss.messaging:service=PostOffice
  </attribute>

  <!-- The default Dead Letter Queue (DLQ) to use for destinations.
       This can be overridden on a per destinatin basis -->
```

```
<attribute name="DefaultDLQ">
  jboss.messaging.destination:service=Queue,name=DLQ
</attribute>

<!--The default maximum number of times to attempt delivery of a
    message before sending to the DLQ (if configured).
    This can be overridden on a per destination basis-->

<attribute name="DefaultMaxDeliveryAttempts">10</attribute>

<!--The default Expiry Queue to use for destinations. This can
    be overridden on a per destinatin basis-->

<attribute name="DefaultExpiryQueue">
  jboss.messaging.destination:service=Queue,name=ExpiryQueue
</attribute>

<!--The default redelivery delay to impose. This can be overridden
    on a per destination basis -->

<attribute name="DefaultRedeliveryDelay">0</attribute>

<!--The periodicity of the message counter manager enquiring on
    queues for statistics-->

<attribute name="MessageCounterSamplePeriod">5000</attribute>

<!--The maximum amount of time for a client to wait for failover
    to start on the server side after it has detected failure-->

<attribute name="FailoverStartTimeout">60000</attribute>

<!--The maximum amount of time for a client to wait for failover
    to complete on the server side after it has detected failure-->

<attribute name="FailoverCompleteTimeout">300000</attribute>

<attribute name="StrictTck">false</attribute>

<!--The maximum number of days results to maintain in the message
    counter history-->

<attribute name="DefaultMessageCounterHistoryDayLimit">-1</attribute>

<!--The name of the connection factory to use for creating
    connections between nodes to pull messages-->

<attribute name="ClusterPullConnectionFactoryName">
  jboss.messaging.connectionfactory:service=ClusterPullConnectionFactory
</attribute>

<!--When redistributing messages in the cluster. Do we need to
    preserve the order of messages received
    by a particular consumer from a particular producer? -->

<attribute name="DefaultPreserveOrdering">false</attribute>

<!-- Max. time to hold previously delivered messages back waiting for
     clients to reconnect after failover -->

<attribute name="RecoverDeliveriesTimeout">300000</attribute>

<!-- Set to true to enable message counters that can be viewed via JMX -->

<attribute name="EnableMessageCounters">false</attribute>
```

```
  <!-- The password used by the message sucker connections to create connections.
       THIS SHOULD ALWAYS BE CHANGED AT INSTALL TIME TO SECURE SYSTEM
  <attribute name="SuckerPassword"></attribute>
  -->

  <!-- The name of the server aspects configuration resource
  <attribute name="ServerAopConfig">aop/jboss-aop-messaging-server.xml</attribute>
  -->
  <!-- The name of the client aspects configuration resource
    <attribute name="ClientAopConfig">aop/jboss-aop-messaging-client.xml</attribute>
  -->

  <depends optional-attribute-name="PersistenceManager">
    jboss.messaging:service=PersistenceManager
  </depends>

  <depends optional-attribute-name="JMSUserManager">
    jboss.messaging:service=JMSUserManager
  </depends>

  <depends>jboss.messaging:service=Connector,transport=bisocket</depends>
  <depends optional-attribute-name="SecurityStore"
    proxy-type="org.jboss.jms.server.SecurityStore">
      jboss.messaging:service=SecurityStore
  </depends>
</mbean>
```

5.2. ServerPeer attributes

This section discusses the **`ServerPeer`** managed bean attributes.

StopServerPeerOnDBFailure
: When set to **`true`**, the server will stop if a database error occurs - e.g. database connection is lost - allowing other nodes to continue work. The default value is **`false`**.

ServerPeerID
: The unique identifier of the **`ServerPeer`**. Each node deployed must have a unique identifier, whether the nodes form a cluster or are linked by a message bridge. The identifier must be a valid integer.

DefaultQueueJNDIContext
: The default JNDI context to be used when binding queues. The default value is **`/queue`**.

DefaultTopicJNDIContext
: The default JNDI context to be used when binding topics. The default value is **`/topic`**.

PostOffice
: The post office used by the **`ServerPeer`**. You will not normally need to edit this attribute. The post office routes messages to queues and maintains the mapping between queues and addresses.

DefaultDLQ
: The default DLQ (Dead Letter Queue) that the server uses for destinations. You can override the DLQ on a per-destination basis. For more information about destinations, refer to *Section 5.9, "Configuring Destinations"*. A DLQ is a destination for messages that the server has failed to deliver more than a certain number of times. If the DLQ is not specified, the message will be removed after the maximum number of delivery attempts. You can specify a global default for the maximum number of delivery attempts with the `DefaultMaxDeliveryAttempts` attribute, or set the maximum individually on a per-destination basis.

DefaultMaxDeliveryAttempts
: The global default for the maximum number of times delivery will be attempted for a message before the message is removed or sent to the DLQ, if configured. The default value is **`10`**. You can override this value on a per-destination basis.

DefaultExpiryQueue
: The default expiry queue that the **`ServerPeer`** will use for destinations. You can override this value on a per-destination basis, as seen in the section on destination managed bean configuration. An expiry queue holds messages that have expired. Message expiry is determined by the value of **`Message::getJMSExpiration()`**. If the expiry queue is not specified, the message will be deleted when it expires.

DefaultRedeliveryDelay
: This attribute lets you delay a redelivery attempt, which helps to prevent thrashing delivery-failure. The default value is **`0`** (that is, no delay). You can override this value on a per-destination basis.

MessageCounterSamplePeriod
: This attribute defines the period of time between the server's queries to the queue for queue statistics. The default value is **`5000`** milliseconds.

FailoverStartTimeout
: The longest period (in milliseconds) that the client will wait for failover to begin on the server side when a problem is detected. The default value is **`60000`** (one minute).

FailoverCompleteTimeout
: The longest period (in milliseconds) that the client will wait for failover to complete on the server side once failover has been initiated. The default value is **`300000`** (five minutes).

DefaultMessageCounterHistoryDayLimit
: JBoss Messaging provides a message counter history, which shows the number of messages arriving on each queue over a certain number of days. This attribute represents the maximum number of days for which to store message counter history. You can override this value on a per-destination.

ClusterPullConnectionFactoryName
: The connection factory used to pull, or suck, messages between queues. You can omit this attribute to disable message sucking while retaining failover.

DefaultPreserveOrdering
: When **`true`**, JMS ordering is preserved in the cluster. See *Chapter 6, Clustering Notes* for more detail. The default value is **`false`**.

RecoverDeliveriesTimeout
: When failover occurs, messages that have been delivered will be stored while the clients reconnect. If the clients do not reconnect (for example, if the client is dead), these messages will eventually time out and be added to the queue. This attribute sets the period before timeout in milliseconds. The default value is **`300000`** (five minutes).

EnableMessageCounters
: When set to **`true`**, enables message counters upon server start.

SuckerPassword
: JBoss Messaging internally creates connections between nodes to redistribute messages between clustered destinations. These connections are created with a special, reserved username. This attribute defines the password to use when creating these connections.

For versions of JBoss Messaging later than 1.4.1.GA, you must define the `SuckerPassword` on the **`SecurityMetadataStore`**.

Warning

The `SuckerPassword` must be changed at install time, or the default password will be used, giving any user who knows the default password access to any destination on the server.

SuckerConnectionRetryTimes
: This is the maximum number of times a sucker's connection is permitted to retry in the event of a failure. The default value is **`-1`** which represents "retry indefinitely".

SuckerConnectionRetryInterval
: This is the interval in milliseconds between each retry of the failed sucker's connection. The default value is **`5000`**.

StrictTck
: To enable strict JMS Technology Compatibility Kit (TCK) semantics, set this attribute to **`true`**.

Destinations
: Returns a list of the destinations (queues and topics) currently deployed.

MessageCounters
: A message counter for a particular queue.

MessageStatistics
: Statistics about each message counter for each queue.

SupportsFailover
: When this attribute is **`false`**, server-side failover does not occur when a node crashes in a cluster.

PersistenceManager
: The persistence manager used by the **`ServerPeer`**. (You will not normally need to change this attribute.)

JMSUserManager
: The JMS user manager used by the **`ServerPeer`**. (You will not normally need to change this attribute.)

SecurityStore
: The pluggable **`SecurityStore`**. If you redefine this attribute, remember that you will need to authenticate the **`MessageSucker`** user (**`JBM.SUCKER`**) with all special permissions required by clustering.

SupportsTxAge
: Specifies whether the transaction creation time is stored in the transaction record. If set to **`true`**, the transaction record is stored. The default is **`false`**.

5.3. ServerPeer methods

The following methods are available for the ServerPeer managed bean:

deployQueue
: Used to programmatically deploy a queue. If the queue exists but is undeployed, it will be deployed. Otherwise, it is created and deployed.

  The `name` parameter matches a destination to deploy.

  The optional `jndiName` parameter represents the full JNDI name of the location to which a destination will be bound. If this is not specified, the destination will be bound in **`<DefaultQueueJNDIContext>/<name>`**.

  There are two overloaded versions of this operation. The first deploys the destination with default paging parameters. The second deploys the destination with the paging parameters specified. For more information about paging parameters, refer to *Section 5.9, "Configuring Destinations"*.

undeployQueue
: Used to programmatically undeploy a queue. Queues are not removed from persistent storage. This operation returns **`true`** if the queue is successfully undeployed. Otherwise, it returns **`false`**.

destroyQueue
: Used to programmatically destroy a queue. Queues are undeployed and all of their data is removed from the database and destroyed.

  Warning

  Exercise caution when using this method, since it will delete all data for the queue.

  This operation returns **`true`** if the queue was destroyed successfully. Otherwise, it returns **`false`**.

deployTopic
: Used to programmatically deploy a topic. There are two overloaded versions of this operation. The first deploys already existing topics with the default paging parameters. The second creates and deploys topics with specified paging parameters. For more information about destinations, refer to *Section 5.9, "Configuring Destinations"*.

  The `name` parameter represents the name of the destination to deploy.

  The `jndiName` represents the full JNDI name of the location to which the destination will be bound. If this is not specified, the destination will be bound in **`<DefaultTopicJNDIContext>/<name>`**.

undeployTopic
: Used to programmatically undeploy a topic. Topics are undeployed, but not removed from persistent storage. This operation returns **`true`** if the topic is undeployed successfully. Otherwise, **`false`** is returned.

destroyTopic
: Used to programmatically destroy a topic. Topics are undeployed and all data is removed from the database and destroyed. This operation returns **`true`** if the topic is successfully destroyed. Otherwise, it returns **`false`**.

Warning

Exercise caution when using this method: it will delete all data for the topic.

listMessageCountersHTML
: Returns message counters in a simply-displayed HTML format.

resetAllMesageCounters
: Resets all message counters to zero.

enableMessageCounters
: Enables all message counters for all destinations. Message counters are disabled by default.

disableMessageCounters
: Disables all message counters for all destinations. Message counters are disabled by default.

retrievePreparedTransactions
: Retrieves a list of the XIDs for all transactions currently in a prepared state on the node.

showPreparedTransactions
: Retrieves a list of the XIDs for all transactions currently in a prepared state on the node in an easily-displayed HTML format.

listAllPreparedTransactions
: Displays the details of all prepared transactions.

listPreparedTransactions
: Displays the details of all prepared transactions where the transaction ages are equal to or older than a specified time.

showMessageDetails
: Displays the details of a message. The message ID is used to specify the message to display.

commitPreparedTransaction
: Manually commit a prepared transaction. The transaction ID is used to specify the transaction to commit.

rollbackPreparedTransaction
: Manually roll-back a prepared transaction. The transaction ID is used to specify the transaction to roll-back.

5.4. Changing the Database

JMS uses the database defined in **`jboss-as/server/$PROFILE/deploy/messaging/<DATABASE_TYPE>-persistence-service.xml`** as a persistence storage. The default persistence storage type is Hypersonic (HSQLDB) defined in **`hsqldb-persistence-service.xml`**. Note that the configuration is defined for a clustered environment (**`<attribute name="Clustered">true</attribute>`**).

Do not Use Hypersonic in Production

Although Hypersonic configuration is used as the default persistence configuration, *Hypersonic is not supported in production* due to the following known issues:

- no transaction isolation
- thread and socket leaks (`connection.close()` does not tidy up resources)
- persistence quality (logs commonly become corrupted after a failure, preventing automatic recovery)
- database corruption
- stability under load (database processes cease when dealing with too much data)
- not viable in clustered environments

The Hypersonic database is intended for developing and testing purposes and should not be used in a production environment. For more information about recommended databases, refer to the *Using Other Databases* chapter in the *Getting Started Guide*.

The Persistence Manager, Post Office and JMS User Manager all interact with persistent storage. The Persistence Manager handles message-related persistence. The Post Office handles binding related persistence. The JMS User Manager handles user-related persistence. All configuration for these managed beans is handled in the **`<your database type>-persistence-service.xml`** file.

Example configuration files for MySQL, Oracle, PostgreSQL, Microsoft SQL Server or Sybase databases are available in the **`jboss-as/docs/examples/jms`** directory of the release bundle.

To enable your database, replace the default **`jboss-as/server/$PROFILE/deploy/messaging/hsqldb-persistence-service.xml`** configuration file with the configuration file for your database type. Restart the server to apply the new persistence configuration.

Clustered Environment not Enabled by Default

Apart from the unsupported Hypersonic configuration, the delivered example configurations are by default set for non-clustered environments. To allow clustered environments for your database configuration, set the Clustered attribute to **`true`** in ***`<your database type>-persistence-service.xml`*** (the default setting is <attribute name="Clustered">false</attribute>).

By default, the messaging services rely on a data store reference **`java:/DefaultDS`** for the data source. To deploy a data source with a different JNDI name, you must update all **`DataSource`** attributes in the persistence configuration file. Example data source configurations are included in the distribution.

5.5. Configuring the Post Office

The post office routes messages to their destination or destinations. It maintains the mappings between the addresses to which a message can be sent, and the final queue. For example, when sending a message with an address that represents a JMS queue, the post office routes the message to that JMS queue. When sending a message with an address that represents a JMS topic, the post office routes the message to a set of queues — one for each JMS subscription.

The post office also handles the persistence for address mapping.

JBoss Messaging post offices are cluster-aware. In a cluster, they automatically route (push) and pull messages between nodes in order to provide fully-distributed JMS queues and topics.

Configure the post office in the **`<database type>-persistence-service.xml`** file. For example:

```
<mbean code="org.jboss.messaging.core.jmx.MessagingPostOfficeService"
  name="jboss.messaging:service=PostOffice"
  xmbean-dd="xmdesc/MessagingPostOffice-xmbean.xml">

  <depends optional-attribute-name="ServerPeer">
    jboss.messaging:service=ServerPeer
  </depends>

  <depends>
    jboss.jca:service=DataSourceBinding,name=DefaultDS
  </depends>

  <depends optional-attribute-name="TransactionManager">
    jboss:service=TransactionManager
  </depends>

  <!-- The name of the post office -->

  <attribute name="PostOfficeName">JMS post office</attribute>

  <!-- The datasource used by the post office to access it's
       binding information -->

  <attribute name="DataSource">java:/DefaultDS</attribute>

  <!-- If true will attempt to create tables and indexes on
       every start-up -->

  <attribute name="CreateTablesOnStartup">true</attribute>

  <!-- If true then we will automatically detect and reject
       duplicate messages sent during failover -->

  <attribute name="DetectDuplicates">true</attribute>

  <!-- The size of the id cache to use when detecting duplicate
       messages -->

  <attribute name="IDCacheSize">500</attribute>

  <attribute name="SqlProperties">
    CREATE_POSTOFFICE_TABLE=CREATE TABLE JBM_POSTOFFICE
      (POSTOFFICE_NAME VARCHAR(255),
      NODE_ID INTEGER, QUEUE_NAME VARCHAR(255), COND VARCHAR(1023),
      SELECTOR VARCHAR(1023), CHANNEL_ID BIGINT, CLUSTERED CHAR(1),
      ALL_NODES CHAR(1),
      PRIMARY KEY(POSTOFFICE_NAME, NODE_ID, QUEUE_NAME)) ENGINE = INNODB

    INSERT_BINDING=INSERT INTO JBM_POSTOFFICE
```

```
      (POSTOFFICE_NAME, NODE_ID, QUEUE_NAME, COND, SELECTOR,
       CHANNEL_ID, CLUSTERED, ALL_NODES)
      VALUES (?, ?, ?, ?, ?, ?, ?, ?)

    DELETE_BINDING=DELETE FROM JBM_POSTOFFICE WHERE
      POSTOFFICE_NAME=? AND NODE_ID=? AND QUEUE_NAME=?

    LOAD_BINDINGS=SELECT QUEUE_NAME, COND, SELECTOR,
      CHANNEL_ID, CLUSTERED, ALL_NODES FROM
   JBM_POSTOFFICE WHERE POSTOFFICE_NAME=? AND NODE_ID=?
  </attribute>

  <!-- This post office is clustered. If you don't want a clustered post
       office then set to false -->

  <attribute name="Clustered">true</attribute>

  <!-- All the remaining properties only have to be specified if the post
       office is clustered. You can safely comment them out if your post
       office is non clustered -->

  <!-- The JGroups group name that the post office will use -->

  <attribute name="GroupName">
    ${jboss.messaging.groupname:MessagingPostOffice}
  </attribute>

  <!-- Max time to wait for state to arrive when the post office
       joins the cluster -->

  <attribute name="StateTimeout">5000</attribute>

  <!-- Max time to wait for a synchronous call to node members using
       the MessageDispatcher -->

  <attribute name="CastTimeout">50000</attribute>

  <!-- Set this to true if you want failover of connections to occur
       when a node is shut down -->

  <attribute name="FailoverOnNodeLeave">false</attribute>

  <!-- JGroups stack configuration for the data channel - used for sending
       data across the cluster -->

  <!-- By default we use the TCP stack for data -->
  <attribute name="DataChannelConfig">
    <config>
      <TCP start_port="7900"
        loopback="true"
        recv_buf_size="20000000"
        send_buf_size="640000"
        discard_incompatible_packets="true"
        max_bundle_size="64000"
        max_bundle_timeout="30"
        use_incoming_packet_handler="true"
        use_outgoing_packet_handler="false"
        down_thread="false" up_thread="false"
        enable_bundling="false"
        use_send_queues="false"
        sock_conn_timeout="300"
        skip_suspected_members="true"/>
      <MPING timeout="4000"
        bind_to_all_interfaces="true"
        mcast_addr="${jboss.messaging.datachanneludpaddress:228.6.6.6}"
        mcast_port="${jboss.messaging.datachanneludpport:45567}"
        ip_ttl="8"
```

```
        num_initial_members="2"
        num_ping_requests="1"/>
      <MERGE2 max_interval="100000"
        down_thread="false" up_thread="false" min_interval="20000"/>
      <FD_SOCK down_thread="false" up_thread="false"/>
      <VERIFY_SUSPECT timeout="1500" down_thread="false"
       up_thread="false"/>
      <pbcast.NAKACK max_xmit_size="60000"
        use_mcast_xmit="false" gc_lag="0"
        retransmit_timeout="300,600,1200,2400,4800"
        down_thread="false" up_thread="false"
        discard_delivered_msgs="true"/>
      <pbcast.STABLE stability_delay="1000" desired_avg_gossip="50000"
        down_thread="false" up_thread="false"
        max_bytes="400000"/>
      <pbcast.GMS print_local_addr="true" join_timeout="3000"
        down_thread="false" up_thread="false"
        join_retry_timeout="2000" shun="false"
        view_bundling="true"/>
    </config>
  </attribute>

  <!-- JGroups stack configuration to use for
       the control channel - used for control messages -->

  <!-- We use udp stack for the control channel -->
  <attribute name="ControlChannelConfig">
    <config>
      <UDP
        mcast_addr="${jboss.messaging.controlchanneludpaddress:228.7.7.7}"
        mcast_port="${jboss.messaging.controlchanneludpport:45568}"
        tos="8"
        ucast_recv_buf_size="20000000"
        ucast_send_buf_size="640000"
        mcast_recv_buf_size="25000000"
        mcast_send_buf_size="640000"
        loopback="false"
        discard_incompatible_packets="true"
        max_bundle_size="64000"
        max_bundle_timeout="30"
        use_incoming_packet_handler="true"
        use_outgoing_packet_handler="false"
        ip_ttl="2"
        down_thread="false" up_thread="false"
        enable_bundling="false"/>
      <PING timeout="2000"
        down_thread="false" up_thread="false" num_initial_members="3"/>
      <MERGE2 max_interval="100000"
        down_thread="false" up_thread="false" min_interval="20000"/>
      <FD_SOCK down_thread="false" up_thread="false"/>
      <FD timeout="10000" max_tries="5" down_thread="false"
        up_thread="false" shun="true"/>
      <VERIFY_SUSPECT timeout="1500" down_thread="false"
       up_thread="false"/>
      <pbcast.NAKACK max_xmit_size="60000"
        use_mcast_xmit="false" gc_lag="0"
        retransmit_timeout="300,600,1200,2400,4800"
        down_thread="false" up_thread="false"
        discard_delivered_msgs="true"/>
      <UNICAST timeout="300,600,1200,2400,3600"
        down_thread="false" up_thread="false"/>
      <pbcast.STABLE stability_delay="1000" desired_avg_gossip="50000"
        down_thread="false" up_thread="false"
        max_bytes="400000"/>
      <pbcast.GMS print_local_addr="true" join_timeout="3000"
        use_flush="true" flush_timeout="3000"
        down_thread="false" up_thread="false"
```

```
        join_retry_timeout="2000" shun="false"
        view_bundling="true"/>
      <FRAG2 frag_size="60000" down_thread="false" up_thread="false"/>
      <pbcast.STATE_TRANSFER down_thread="false" up_thread="false"
        use_flush="true" flush_timeout="3000"/>
      <pbcast.FLUSH down_thread="false" up_thread="false" timeout="20000"
        auto_flush_conf="false"/>
    </config>
  </attribute>

</mbean>
```

5.5.1. MessagingPostOffice Attributes

`MessagingPostOffice` Service Attributes are described in the following list.

DataSource
: The datasource the postoffice should use for persisting its mapping data.

SQLProperties
: This is where the DDL and DML for the particular database is specified. If a particular DDL or DML statement is not overridden, the default Hypersonic configuration will be used for that statement.

CreateTablesOnStartup
: Set this to **`true`** if you wish the post office to attempt to create the tables (and indexes) when it starts. If the tables (or indexes) already exist a **`SQLException`** will be thrown by the JDBC driver and ignored by the Persistence Manager, allowing it to continue.

  By default the value of **`CreateTablesOnStartup`** attribute is set to **`true`**

DetectDuplicates
: Set this to **`true`** if you wish the post office detect duplicate messages that may sent when a send is retried on a different node after server failure.

  By default the value of **`DetectDuplicates`** attribute is set to **`true`**

IDCacheSize
: If duplicate detection is enabled. (See **`DetectDuplicates`**), then the server will remember the last **`n`** message ids sent, to prevent duplicate messages sent after failover has occurred. The value of **`n`** is determined by this attribute.

  By default the value of **`IDCacheSize`** attribute is set to **`500`**

PostOfficeName
: The name of the post office.

NodeIDView
: This returns set containing the node ids of all the nodes in the cluster.

GroupName
: All post offices in the cluster with the same group name will form a cluster together. Make sure the group name matches with all the nodes in the cluster you want to form a cluster with.

Clustered
: If true the post office will take part in a cluster to form distributed queues and topics. If false then it will not participate in the cluster. If false, then all the cluster related attributes will be ignored.

StateTimeout
: The maximum time to wait when waiting for the group state to arrive when a node joins a pre-existing cluster.

  The default value is **`5000`** milliseconds.

CastTimeout
: The maximum time to wait for a reply casting message synchronously.

  The default value is **`5000`** milliseconds.

FailoverOnNodeLeave
: Specifies how messages stored on a node are redistributed when a node is cleanly shutdown. The default value is **`false`**. If **`true`**, when a server node is shut down cleanly (using **`Ctrl`**+**`C`** in the terminal) all messages stored on the node are moved to another node in the cluster.

Important

Clients originally connected to the cleanly shutdown node are not automatically reconnected to the failover node in the cluster. Clients return an exception upon message failover.

MaxConcurrentReplications
: The maximum number of concurrent replication requests to make before blocking for replies to come back. This prevents us overwhelming JGroups. This is rarely a good reason to change this.

  The default value is **`50`**

ControlChannelConfig
: JBoss Messaging uses JGroups for all group management. This contains the JGroups stack configuration for the control channel.

  The control channel is used for sending request/receiving responses from other nodes in the cluster

  The details of the JGroups configuration won't be discussed here since it is standard JGroups configuration. Detailed information on JGroups can be found in JGroups release documentation or on-line at *http://www.jgroups.org* or *http://wiki.jboss.org/wiki/Wiki.jsp?page=JGroups*.

DataChannelConfig
: JBoss Messaging uses JGroups for all group management. This contains the JGroups stack configuration for the data channel.

  The data channel is used for sending sending/receiving messages from other nodes in the cluster and for replicating session data.

  The details of the JGroups configuration won't be discussed here since it is standard JGroups configuration. Detailed information on JGroups can be found in JGroups release documentation or on-line at *http://www.jgroups.org* or *http://wiki.jboss.org/wiki/Wiki.jsp?page=JGroups*.

Database Connection Retry Parameters control whether a connection should be re-established when a connection failure is detected, how many attempts to reconnect should be made, and at what interval each attempt should be made.

RetryOnConnectionFailure
: Specifies whether the MBean should attempt to reconnect to the database. The default is **`false`**.

MaxRetry
: Specifies the maximum limit for DataSource connection failures. The default is **`25`**.
Set the parameter to **`-1`** to activate "retry forever" mode. This parameter is valid when *`RetryOnConnectionFailure`* is set to **`true`**.

Important - Closing Clients Issue on Nodes with MaxRetry set to -1

Clients that consume messages from a clustered destination can become unresponsive when closed. This happens if the node has the value *`MaxRetry`* set to **`-1`**, and loses the connection to the database. To avoid the problem, set the node parameter *`MaxRetry`* to a value greater than **`-1`**. You can set the attribute value in the MBeans **`PersistenceManager`**, **`PostOffice`**, and **`JMSUserManager`** in the file **`[database]-persistence-service.xml`**

RetryInterval
: Specifies the retry interval between two consecutive retries. The default is 1000 (milliseconds).
This parameter is valid when *`RetryOnConnectionFailure`* is set to **`true`**.

If a node becomes unresponsive for a few seconds while still being stable (for example during a minor network malfunction), past failover behavior reported the node had left the cluster. This scenario can be avoided by specifying the following parameters in the MessagePostOffice bean. The following parameters provide the MessagePostOffice access to the cluster timestamp table, which is used to more accurately determine the state of a node.

Important

The timestamp table parameters should be used in conjunction with the MessagingClusterHealthMBean MBean. You deploy this MBean in the ServerPeer MBean. Refer to *Section 5.1, "Configuring the ServerPeer"* for a full overview of the supported parameters and operations.

KeepOldFailoverMode
: Specifies whether the timestamp table failover mode should be used. The default is **`true`** (disable new failover behavior).

NodeStateRefreshInterval
: Specifies the maximum duration (in milliseconds) a cluster will wait for a node to refresh its timestamp before the cluster marks the node as disabled. The default is 30000 (30 seconds).

5.6. Configure the MessagingClusterHealthMBean

The **`MessagingPostOffice`** uses the *`KeepOldFailoverMode`* and *`NodeStateRefreshInterval`* parameters to control the duration a node must update timestamps to

remain part of a cluster. These parameters do not control how a node responds to losing connection to the database.

If a node loses connection to the database, it can not update the timestamp information available to the **`MessagePostOffice`**. Even though the node may still be healthy, the cluster has no way of determining the actual state of the node. As a result, the cluster assumes the node has failed and duplicate message delivery can occur.

The **`MessagingClusterHealthMBean`** MBean is responsible for monitoring the node state, and stopping and starting a node when it loses connection to the database. When a node is shunned from the cluster and subsequently loses database connectivity, the MBean shuts down the node immediately. The MBean monitors the JGroups status and database status while the node is unavailable, and will restart the node when it detects JGroups and the database connection is restored to normal operation.

To enable this functionality, declare the **`MessagingClusterHealthMBean`** MBean in the ServerPeer MBean as an *`optional-attribute-name`* attribute in a <depends> directive.

```
<!-- ServerPeer MBean configuration ============================== -->
<mbean code="org.jboss.jms.server.ServerPeer"       name="jboss.messaging:service=ServerPeer"
      xmbean-dd="xmdesc/ServerPeer-xmbean.xml">

<!-- The unique id of the server peer - in a cluster each node MUST have a unique value -
 must be an integer -->

   <attribute name="ServerPeerID">0</attribute>

<!-- The default JNDI context to use for queues when they are deployed without specifying one
 -->

   <attribute name="DefaultQueueJNDIContext">/queue</attribute>

<!-- The default JNDI context to use for topics when they are deployed without specifying one
 -->

   <attribute name="DefaultTopicJNDIContext">/topic</attribute>

<!-- XML CONFIG REMOVED FOR READABILITY -->


   <depends optional-attribute-name="PersistenceManager">
      jboss.messaging:service=PersistenceManager
   </depends>

<!-- XML CONFIG REMOVED FOR READABILITY -->

   <depends optional-attributename="MessagingClusterHealthMBean">
      jboss.messaging:service=MessagingClusterHealthMBean
   </depends>

</mbean>
```

5.7. Configuring the Persistence Manager

JBoss Messaging ships with a JDBC Persistence Manager, which handles message data persistence in a relational database accessed via JDBC. The Persistence Manager can be plugged into the Messaging server, which allows additional implementations to persist message data in non-relational stores, and file stores.

Persistent service configuration details are grouped in **`<database type>-persistence-service.xml`**. JBoss Messaging ships with the **`hsqldb-persistence-service.xml`** file by

default, which configures the Messaging server to use the Hypersonic database instance included by default with any JBoss Enterprise Application Server instance.

Warning

Hypersonic is not supported for use in a production environment.

JBoss Messaging also ships with Persistence Manager configurations for MySQL, Oracle, PostgreSQL, Sybase, Microsoft SQL Server, and DB2. The example configuration files (such as **`mysql-persistence-service.xml`** and **`ndb-persistence-service.xml`**) are available from the **`jboss-as/docs/examples/jms`** directory of the release bundle.

The JDBC Persistence Manager uses standard SQL as its Data Manipulation Language (DML), so writing a Persistence Manager configuration for another database type is a matter of changing the configuration's Data Definition Language (DDL), which usually differs on a per-database basis.

JBoss Messaging also ships with a Null Persistence Manager configuration option, which can be used when persistence is not required.

The following code is the default Hypersonic persistence manager configuration:

```
<mbean code="org.jboss.messaging.core.jmx.JDBCPersistenceManagerService"
  name="jboss.messaging:service=PersistenceManager"
  xmbean-dd="xmdesc/JDBCPersistenceManager-xmbean.xml">

  <depends>jboss.jca:service=DataSourceBinding,name=DefaultDS</depends>

  <depends optional-attribute-name="TransactionManager">
    jboss:service=TransactionManager
  </depends>

  <!-- The datasource to use for the persistence manager -->

  <attribute name="DataSource">java:/DefaultDS</attribute>

  <!-- If true will attempt to create tables and indexes on every start-up -->

  <attribute name="CreateTablesOnStartup">true</attribute>

  <!-- If true then will use JDBC batch updates -->

  <attribute name="UsingBatchUpdates">false</attribute>

  <!-- The maximum number of parameters to include in a prepared statement -->

  <attribute name="MaxParams">500</attribute>
</mbean>
```

Important

The maximum size of Sybase database text and image data types is set to **`2 kilobytes`** by default. Any message that exceeds this limit is truncated, without any information or warning. Set the `@@TEXTSIZE` database parameter to a higher value to prevent potential truncation.

Truncation may also occur in the Microsoft SQL Server if `@@TEXTSIZE` value is set to a lesser value than the default value. For further information, refer to *https://issues.jboss.org/browse/SOA-554*, and the *Special Notes on Sybase* section in the *Administration and Configuration Guide*.

Important

Microsoft SQL Server does not automatically unallocate hard drive space when data is deleted from a database. When the hard drive database space is used as a data store for a service that temporarily stores many records (such as a messaging service), the disk space will quickly become much greater than the amount of data actually being stored.

Database administrators must implement database maintenance plans to ensure that the unused space is reclaimed. Refer to your Microsoft SQL Server documentation for the DBCC commands **`ShrinkDatabase`** and **`UpdateUsage`** for guidance reclaiming the unused space. For further information about this issue, refer to *https://issues.jboss.org/browse/SOA-629*

5.7.1. JDBCPersistenceManager MBean Attributes

The JDBCPersistenceManager attributes are described in the following list.

CreateTablesOnStartup
: Set this to **`true`** if you wish the Persistence Manager to attempt to create the tables (and indexes) when it starts. If the tables (or indexes) already exist a **`SQLException`** will be thrown by the JDBC driver and ignored by the Persistence Manager, allowing it to continue.

  By default the value of **`CreateTablesOnStartup`** attribute is set to **`true`**.

UsingBatchUpdates
: Set this to **`true`** if the database supports JDBC batch updates. The JDBC Persistence Manager will then group multiple database updates in batches to aid performance.

  By default the value of **`UsingBatchUpdates`** attribute is set to **`false`**

UsingBinaryStream
: Set this to **`true`** if you want messages to be store and read using a JDBC binary stream rather than using getBytes(), setBytes(). Some database has limits on the maximum number of bytes that can be get/set using getBytes()/setBytes().

  By default the value of **`UsingBinaryStream`** attribute is set to **`true`**

UsingTrailingByte
: Certain version of Sybase are known to truncate blobs if they have trailing zeros. To prevent this if this attribute is set to **`true`** then a trailing non zero byte will be added and removed to each blob before and after persistence to prevent the database from truncating it. Currently this is only known to be necessary for Sybase.

  By default the value of **`UsingTrailingByte`** attribute is set to **`false`**

SupportsBlobOnSelect
: Oracle (and possibly other databases) is known to not allow BLOBs to be inserted using a INSERT INTO ... SELECT FROM statement, and requires a two stage conditional insert of messages. If this value is false then such a two stage insert will be used.

  By default the value of **`SupportsBlobOnSelect`** attribute is set to **`true`**

SQLProperties
: This is where the DDL and DML for the particular database is specified. If a particular DDL or DML statement is not overridden, the default Hypersonic configuration will be used for that statement.

MaxParams
: When loading messages the persistence manager will generate prepared statements with many parameters. This value tells the persistence manager what the absolute maximum number of parameters are allowable per prepared statement.

  By default the value of **`MaxParams`** attribute is set to **`100`**

UseNDBFailoverStrategy
: When running in a clustered database environment it is possible that some databases, MySQL for instance, can fail during the commit of a database transaction. This can happen if the database node dies whilst committing meaning that the final state of the transaction is unknown. If this attribute is set to true and the above happens then the SQL statement will be re-executed, however if there is a further error an assumption is made that this is because the previous transaction committed successfully and the error is ignored.

  By default the value of **`UseNDBFailoverStrategy`** attribute is set to **`false`**

Database Connection Retry Parameters control whether a connection should be re-established when a connection failure is detected, how many attempts to reconnect should be made, and at what interval each attempt should be made.

RetryOnConnectionFailure
: Specifies whether the MBean should attempt to reconnect to the database. The default is **`false`**.

MaxRetry
: Specifies the maximum limit for DataSource connection failures. The default is **`25`**.
  Set the parameter to **`-1`** to activate "retry forever" mode. This parameter is valid when *`RetryOnConnectionFailure`* is set to **`true`**.

Important - Closing Clients Issue on Nodes with MaxRetry set to -1

Clients that consume messages from a clustered destination can become unresponsive when closed. This happens if the node has the value *`MaxRetry`* set to **`-1`**, and loses the connection to the database. To avoid the problem, set the node parameter *`MaxRetry`* to a value greater than **`-1`**. You can set the attribute value in the MBeans **`PersistenceManager`**, **`PostOffice`**, and **`JMSUserManager`** in the file **`[database]-persistence-service.xml`**

RetryInterval
: Specifies the retry interval between two consecutive retries. The default is 1000 (milliseconds). This parameter is valid when *`RetryOnConnectionFailure`* is set to **`true`**.

CreateTablesOnStartup
: Specifies whether tables and index creation is attempted when the Persistence Manager is started. When set to **`true`** (default), the persistence manager will attempt to create tables (and indexes) on startup. If tables or indexes already exist, a `SQLException` will be thrown by the JDBC driver and ignored by the persistence manager, allowing it to continue unhindered.

UsingBatchUpdates
: Specifies whether multiple database updates are grouped in batches to improve performance. Set this value to **`true`** if your database supports JDBC batch updates.. The default value is **`false`**.

UsingBinaryStream
: Specifies whether messages are stored and read with a JDBC binary stream, instead of via **`getBytes()`** and **`setBytes()`**. Set this value to **`false`** if your database must use **`getBytes()`** and **`setBytes()`**. The default value is **`true`**.

UsingTrailingByte
: Specifies how Sybase database BLOBs containing trailing zeroes are handled. When set to **`true`** , a trailing non-zero byte is added to each BLOB before persistence, and removed from the BLOB following persistence, preventing truncation by the database. The default value is **`false`**

Note

Certain versions of Sybase truncate a BLOB with trailing zeros. This attribute is only required if you are running a Sybase database.

SupportsBlobOnSelect
: Specifies how BLOBs are inserted into certain database types. When set to **`false`**, two-stage insertion will be used. The default value is **`true`**.

Note

Certain databases, specifically Oracle, do not allow BLOB insertion via an **`INSERT INTO ... SELECT FROM`** statement, and require two-stage conditional message insertion. Set this attribute to **`false`** if you are running an Oracle database, or other database with this requirement.

SQLProperties
: Specifies the DDL and DML for a particular database. If a particular DDL or DML statement is not overridden, the default Hypersonic configuration will be used for that statement.

UseNDBFailoverStrategy
: Specifies whether a SQL statement is re-executed in the event a database transation commit fails in a clustered environment. If set to **`true`**, the SQL statement is re-executed in the event that the commit fails. If a further error occurs, the persistence manager assumes the error is due to the previous transaction having committed successfully, and ignores the error. By default, this attribute is set to **`false`**.

Note

When some databases, such as MySQL, run in clustered environments, they can fail during database transaction commits. If this occurs, the final transaction state is unknown.

MaxParams
: Specifies the maximum number of parameters allowed per prepared statement while loading messages. The default value is **`500`**.

5.8. Configuring the JMS user manager

The JMS User Manager maps pre-configured client IDs to users. It also manages user and role tables, depending on the configured login module.

The following is an example **`JMSUserManager`** configuration:

```
<mbean code="org.jboss.jms.server.plugin.JDBCJMSUserManagerService"
  name="jboss.messaging:service=JMSUserManager"
  xmbean-dd="xmdesc/JMSUserManager-xmbean.xml">
  <depends>jboss.jca:service=DataSourceBinding,name=DefaultDS</depends>
  <depends optional-attribute-name="TransactionManager">
    jboss:service=TransactionManager
  </depends>
  <attribute name="DataSource">java:/DefaultDS</attribute>
  <attribute name="CreateTablesOnStartup">true</attribute>
  <attribute name="SqlProperties">
    CREATE_USER_TABLE=CREATE TABLE JBM_USER (USER_ID VARCHAR(32) NOT NULL,
      PASSWD VARCHAR(32) NOT NULL, CLIENTID VARCHAR(128),
      PRIMARY KEY(USER_ID)) ENGINE = INNODB
```

```
    CREATE_ROLE_TABLE=CREATE TABLE JBM_ROLE (ROLE_ID VARCHAR(32) NOT NULL,
      USER_ID VARCHAR(32) NOT NULL, PRIMARY KEY(USER_ID, ROLE_ID))
      ENGINE = INNODB

    SELECT_PRECONF_CLIENTID=SELECT CLIENTID FROM JBM_USER WHERE USER_ID=?

    POPULATE.TABLES.1=INSERT INTO JBM_USER (USER_ID,PASSWD,CLIENTID)
      VALUES ('jdoe','jdoepw','jdoe-id')
  </attribute>
</mbean>
```

5.8.1. JMSUserManager MBean Attributes

CreateTablesOnStartup

Set this to **`true`** if you wish the JMS user manager to attempt to create the tables (and indexes) when it starts. If the tables (or indexes) already exist a **`SQLException`** will be thrown by the JDBC driver and ignored by the Persistence Manager, allowing it to continue.

By default the value of **`CreateTablesOnStartup`** attribute is set to **`true`**

UsingBatchUpdates

Set this to **`true`** if the database supports JDBC batch updates. The JDBC Persistence Manager will then group multiple database updates in batches to aid performance.

By default the value of **`UsingBatchUpdates`** attribute is set to **`false`**

SQLProperties

This is where the DDL and DML for the particular database is specified. If a particular DDL or DML statement is not overridden, the default Hypersonic configuration will be used for that statement.

Default user and role data can also be specified here. Any data to be inserted must be specified with property names starting with **`POPULATE.TABLES`** as in the above example.

Database Connection Retry Parameters control whether a connection should be re-established when a connection failure is detected, how many attempts to reconnect should be made, and at what interval each attempt should be made.

RetryOnConnectionFailure

Specifies whether the MBean should attempt to reconnect to the database. The default is **`false`**.

MaxRetry

Specifies the maximum limit for DataSource connection failures. The default is **`25`**.
Set the parameter to **`-1`** to activate "retry forever" mode. This parameter is valid when *`RetryOnConnectionFailure`* is set to **`true`**.

Important - Closing Clients Issue on Nodes with MaxRetry set to -1

Clients that consume messages from a clustered destination can become unresponsive when closed. This happens if the node has the value *`MaxRetry`* set to **`-1`**, and loses the connection to the database. To avoid the problem, set the node parameter *`MaxRetry`* to a value greater than **`-1`**. You can set the attribute value in the MBeans **`PersistenceManager`**, **`PostOffice`**, and **`JMSUserManager`** in the file **`[database]-persistence-service.xml`**

RetryInterval
: Specifies the retry interval between two consecutive retries. The default is 1000 (milliseconds). This parameter is valid when *`RetryOnConnectionFailure`* is set to **`true`**.

5.9. Configuring Destinations

5.9.1. Pre-configured destinations

JBoss Messaging ships with a default set of preconfigured destinations that are deployed at server start-up. The configuration information for these destinations can be found in the following section of **`destinations-service.xml`**:

```
<!--
      The Default Dead Letter Queue. This destination is a dependency of an EJB MDB
 container.
   -->

   <mbean code="org.jboss.jms.server.destination.QueueService"
      name="jboss.messaging.destination:service=Queue,name=DLQ"
      xmbean-dd="xmdesc/Queue-xmbean.xml">
      <depends optional-attribute-name="ServerPeer">jboss.messaging:service=ServerPeer</
depends>
      <depends>jboss.messaging:service=PostOffice</depends>
   </mbean>

   <!--
      The Default Expiry Queue.
   -->

   <mbean code="org.jboss.jms.server.destination.QueueService"
      name="jboss.messaging.destination:service=Queue,name=ExpiryQueue"
      xmbean-dd="xmdesc/Queue-xmbean.xml">
      <depends optional-attribute-name="ServerPeer">jboss.messaging:service=ServerPeer</
depends>
      <depends>jboss.messaging:service=PostOffice</depends>
   </mbean>
```

5.9.2. Configuring Queues

5.9.2.1. Queue MBean Attributes

Name
: Defines the queue name.

JNDIName
: Defines the JNDI name that binds the queue.

DLQ
: Defines the DLQ (Dead Letter Queue) for this queue and overrides any value set in the Server Peer configuration file.

ExpiryQueue
: Defines the expiry queue and overrides any value set in the Server Peer configuration file.

RedeliveryDelay
: Defines the redelivery delay to be applied to this queue and overrides any value set in the Server Peer configuration file.

MaxDeliveryAttempts
: Defines the maximum number of times message delivery is attempted before the message is sent to the DLQ, if configured. The default value, **`-1`**, means that the value from the Server Peer configuration file is used. Any other setting will override the value set in the Server Peer configuration file.

CreatedProgrammatically
: Returns **`true`** if the queue was created programmatically.

MessageCount
: Returns the total number of messages in the queue. That is, the number of messages being scheduled plus the number being delivered, plus the number not being delivered.

ScheduledMessageCount
: Returns the number of *scheduled messages* in the queue. This is the number of messages scheduled to be delivered at a later date.

  Scheduled delivery lets you specify the earliest time at which a particular message will be delivered. For example, you can send a message now, and specify that it will not be delivered for two hours. To do so, set the following in the message header:

```
long now = System.currentTimeMillis();

Message msg = sess.createMessage();

msg.setLongProperty(JBossMessage.JMS_JBOSS_SCHEDULED_DELIVERY_PROP_NAME,
  now + 1000 * 60 * 60 * 2);

prod.send(msg);
```

MaxSize
: Specifies the maximum number of messages that can be held in a queue. Any excess messages will be dropped. The default value is **-1**, which is unbounded.

Clustered
: This attribute must be set to **true** if the destination is clustered.

MessageCounter
: Each queue maintains a message counter.

MessageCounterStatistics
: The statistics for the message counter.

MessageCounterHistoryDayLimit
: The maximum number of days for which to hold message counter history. Overrides any value set in the Server Peer configuration file.

ConsumerCount
: The number of consumers currently consuming from the queue.

DropOldMessageOnRedeploy
: Specifies how queue services with clustered attributes that differ from previously deployed attributes are handled. If set to **true**, all remaining messages in the queue are deleted after the queue service re-deployment if the queue service attribute contains a different clustered attribute. If set to **false** (default), all messages are reserved.

Warning - Redeployment Considerations

When you re-deploy a destination, you must shut down all the nodes in the cluster, make proper configuration changes, and then restart the nodes.

Redeploying from a non-clustered to a clustered queue requires you set the clustered attribute to **true**, and add the queue service configuration to each node.

Redeploying from a clustered to a non-clustered queue requires you set the clustered attribute to **false** in one of the queue configurations and delete all other queues in the cluster.

5.9.2.1.1. Destination Security Configuration

<SecurityConfig> determines which roles can read, write and create upon the destination. It uses the same syntax and semantics as JBossMQ destination security configuration.

```
<SecurityConfig>
  <security>
    <role read="true" write="true" create="true"/>
  </security>
<SecurityConfig>
```

The <SecurityConfig> element must contain one <security> element, which can contain multiple <role> elements. A <role> element defines the access type for that particular role using the following attributes:

read
: Specifies the role can create consumers, receive messages, and browse the destination.

write
: Specifies the role can create producers, or send messages to the destination.

create
: Specifies the role can create durable subscriptions on this destination.

Note

Configuring security for a destination is optional. If a **`SecurityConfig`** element is not specified, then the default security configuration from the Server Peer will be used instead.

5.9.2.1.2. Destination paging parameters

Pageable Channels is a JBoss Messaging feature that lets you specify a maximum number of messages to be stored in memory at one time, on a queue-by-queue or topic-by-topic basis. JBoss Messaging then pages messages to and from storage transparently in blocks. This allows queues and subscriptions to grow to very large sizes without any degradation in performance as channel size increases.

The individual parameters associated with pageable channels are as follows:

FullSize
: Specifies the maximum number of messages held by the queue or topic subscription in memory at any one time. The actual queue can hold more messages, but these are paged to and from storage as messages are added or consumed. If no value is specified, the default is **`75000`**.

PageSize
: Specifies the maximum number of messages that are pre-loaded per operation when loading messages from the queue or subscription. If no value is specified, the default is **`2000`**.

DownCacheSize
: Specifies the maximum number of messages the Down Cache holds before the messages are flushed to storage. The default value is **`2000`** messages.

  When messages are paged to storage from the queue, they enter a *Down Cache* before being written to storage. This enables the write to occur as a single operation, which aids performance.

Note

Paging parameters for temporary queues must be specified on the appropriate connection factory. Refer to *Section 5.10, "Configuring Connection Factories"* for detailed information about the different connection factories available.

5.9.2.1.3. Queue Managed Bean Operations

RemoveAllMessages
: Removes (and deletes) all messages from the queue.

Important

This will permanently delete all messages from the queue; use this operation with caution.

ListAllMessages
: Lists all messages currently in the queue. Using a JMS selector as an argument in this operation lets you retrieve a subset of the messages in the queue that match the given criteria.

ListDurableMessages
: Lists all *durable* messages in the queue. Using a JMS selector as an argument in this operation lets you retrieve a subset of messages in the queue that match the given criteria.

ListNonDurableMessages
: Lists all *non-durable* messages in a queue. Using a JMS selector as an argument in this operation lets you retrieve a subset of messages in the queue that match the given criteria.

ResetMessageCounter
: Resets the message counter to zero.

ResetMessageCounterHistory
: Resets the message counter history.

ListMessageCounterAsHTML
: Lists the message counter in HTML format.

ListMessageCounterHistoryAsHTML
: Lists the message counter history in HTML format.

5.9.3. Configuring Topics

5.9.3.1. Topic Managed Bean Attributes

Name
: Defines the name of the topic.

JNDIName
: Defines the JNDI location where the topic is bound.

DLQ
: Defines the Dead Letter Queue (DLQ) used for this topic and overrides any value set in the Server Peer configuration file.

ExpiryQueue
: Defines the expiry queue used for this topic and overrides any value set in the Server Peer configuration file.

RedeliveryDelay
: Defines the delay period between redelivery attempts for this topic and overrides any value set in the Server Peer configuration file.

MaxDeliveryAttempts
: Defines the maximum number of times message delivery will be attempted before the message is sent to the DLQ, if configured. The default value is **`-1`**, which specifies that the value from the Server Peer configuration file be used. Any other setting overrides the Server Peer value.

CreatedProgrammatically
: Returns **`true`** if the topic was created programmatically.

MaxSize
: Specifies the maximum number of messages that can be held in a topic subscription. Any excess messages will be dropped from the topic. The default value is **`-1`**, which applies no size restriction.

Clustered
: Set this to **`true`** if your destination is clustered.

MessageCounterHistoryDayLimit
: Defines the maximum number of days to retain message counter history, and overrides any value set in the Server Peer configuration file.

MessageCounters
: Returns a list of message counters for the topic's subscriptions.

AllMessageCount
: Returns the total number of messages in all subscriptions belonging to the topic.

DurableMessageCount
: Returns the total number of *durable* messages in all subscriptions belonging to this topic.

NonDurableMessageCount
: Returns the total number of *non-durable* messages in all subscriptions belonging to this topic.

DropOldMessageOnRedeploy
: Specifies how queue services with clustered attributes that differ from previously deployed attributes are handled. If set to **`true`**, all remaining messages in the queue are deleted after the queue service re-deployment if the queue service attribute contains a different clustered attribute. If set to **`false`** (default), all messages are reserved.

Warning - Redeployment Considerations

When you re-deploy a destination, you must shut down all the nodes in the cluster, make proper configuration changes, and then restart the nodes.

Redeploying from a non-clustered to a clustered queue requires you set the clustered attribute to **`true`**, and add the queue service configuration to each node.

Redeploying from a clustered to a non-clustered queue requires you set the clustered attribute to **`false`** in one of the queue configurations and delete all other queues in the cluster.

AllSubscriptionsCount
: Returns a count of all subscriptions belonging to this topic.

DurableSubscriptionsCount
: Returns a count of all durable subscriptions belonging to this topic.

NonDurableSubscriptionsCount
: Returns a count of all non-durable subscriptions belonging to this topic.

5.9.3.1.1. Destination Security Configuration

<SecurityConfig> determines which roles can read, write and create upon the destination. It uses the same syntax and semantics as JBossMQ destination security configuration.

The <SecurityConfig> element must contain one <security> element, which can contain multiple <role> elements. A <role> element defines the access type for that particular role using the following attributes:

read
: Specifies the role can create consumers, receive messages, and browse the destination.

write
: Specifies the role can create producers, or send messages to the destination.

create
: Specifies the role can create durable subscriptions on this destination.

Note

Configuring security for a destination is optional. If a **`SecurityConfig`** element is not specified, then the default security configuration from the Server Peer will be used instead.

5.9.3.1.2. Destination paging parameters

Previously, for an application to support a queue or subscription, the queue needed to be stored entirely in memory. This was not always possible for very large queues or subscriptions.

Pageable Channels is a new JBoss Messaging feature that lets you specify a maximum number of messages to be stored in memory at one time, on a queue-by-queue or topic-by-topic basis. JBoss Messaging then pages messages to and from storage transparently in blocks. This allows queues and subscriptions to grow to very large sizes without any degradation in performance as channel size increases. It has been tested with queues in excess of ten million 2 kilobyte messages on very basic hardware, and has the potential to scale to much greater message numbers.

The individual parameters associated with pageable channels are as follows:

FullSize
: Specifies the maximum number of messages held by the queue or topic subscription in memory at any one time. The actual queue can hold more messages, but these are paged to and from storage as messages are added or consumed. If no value is specified, the default is **`75000`**.

PageSize
: Specifies the maximum number of messages that are pre-loaded per operation when loading messages from the queue or subscription. If no value is specified, the default is **`2000`**.

DownCacheSize
: Specifies the maximum number of messages the Down Cache holds before the messages are flushed to storage. The default value is **`2000`** messages.

  When messages are paged to storage from the queue, they enter a *Down Cache* before being written to storage. This enables the write to occur as a single operation, which aids performance.

Note

Paging parameters for temporary queues must be specified on the appropriate connection factory. See the section on Connection Factory Configuration for details.

5.9.3.2. Topic Managed Bean Operations

RemoveAllMessages
: Removes (and deletes) all messages from subscriptions that belong to this topic.

Important

This will permanently delete all messages from the topic; use this operation with caution.

ListAllMessages
: Lists all messages belonging to a specified subscription. Using a JMS selector as an argument in this operation lets you retrieve a subset of messages in the queue that match the given criteria.

ListDurableMessages
: Lists all durable messages belonging to the specified subscription. Using a JMS selector as an argument in this operation lets you retrieve a subset of messages in the queue that match the given criteria.

ResetMessageCounter
: Resets the message counter to zero.

ResetMessageCounterHistory
: Resets the message counter history.

ListAllSubscriptionsAsHTML
: Lists all subscriptions belonging to this topic in HTML format.

ListDurableSubscriptionsAsHTML
: Lists all durable subscriptions belonging to this topic in HTML format.

ListNonDurableSubscriptions
: Lists all non-durable messages belonging to the specified subscription. Using a JMS selector as an argument in this operation lets you retrieve a subset of messages in the queue that match the given criteria.

ListNonDurableSubscriptionsAsHTML
: Lists all non-durable subscriptions belonging to this topic in HTML format.

5.10. Configuring Connection Factories

JBoss Messaging is configured by default to bind two connection factories in JNDI upon startup.

The first connection factory is the default, non-clustered connection factory. This connection factory is provided to maintain compatibility with applications originally written against JBossMQ, which does not include automatic failover or load balancing. If you do not require client-side automatic failover or load balancing, then you should use this first connection factory.

The first connection factory is bound into the following JNDI contexts:

- **`/ConnectionFactory`**
- **`/XAConnectionFactory`**
- **`java:/ConnectionFactory`**
- **`java:/XAConnectionFactory`**.

The second connection factory is the default clustered connection factory, which is bound into the following JNDI contexts:

- **`/ClusteredConnectionFactory`**
- **`/ClusteredXAConnectionFactory`**
- **`java:/ClusteredConnectionFactory`**
- **`java:/ClusteredXAConnectionFactory`**

If you want to provide a default client ID for a connection factory, or bind a connection factory to a different JNDI locationConsider, then configure and deploy additional connection factories. To deploy a new connection factory, configure a new **`ConnectionFactory`** managed bean in **`connection-factories-service.xml`**.

You can also create a new service deployment descriptor, **`<name>-service.xml`**, and deploy it in **`$JBOSS_HOME/server/messaging/deploy`**.

Enable support for automatic failover or load balancing by setting the relevant attributes in your connection factory:

> Example 5.1. Connection Factory
> This example connection factory creates a connection factory with the preconfigured client ID **`myClientID`**, which is bound to two locations in the JNDI tree: **`/MyConnectionFactory`** and **`/factories/cf`**.
>
> The example overrides the following default values:
>
> - `PreFetchSize`
> - `DefaultTempQueueFullSize`
> - `DefaultTempQueuePageSize`
> - `DefaultTempQueueDownCacheSize`

- `DupsOKBatchSize`
- `SupportsFailover`
- `SupportsLoadBalancing`
- `LoadBalancingFactory`

The connection factory uses the default remoting connector. To use a different remoting connector with the connection factory, change the **`Connector`** attribute to specify the service name of the connector you wish to use.

```
<mbean code="org.jboss.jms.server.connectionfactory.ConnectionFactory"
  name="jboss.messaging.connectionfactory:service=MyConnectionFactory"
  xmbean-dd="xmdesc/ConnectionFactory-xmbean.xml">
  <depends optional-attribute-name="ServerPeer">
    jboss.messaging:service=ServerPeer
  </depends>
  <depends optional-attribute-name="Connector">
    jboss.messaging:service=Connector,transport=bisocket
  </depends>
  <depends>jboss.messaging:service=PostOffice</depends>

  <attribute name="JNDIBindings">
    <bindings>
    <binding>/MyConnectionFactory</binding>
    <binding>/factories/cf</binding>
    </bindings>
  </attribute>

  <attribute name="ClientID">myClientID</attribute>

  <attribute name="SupportsFailover">true</attribute>

  <attribute name="SupportsLoadBalancing">false</attribute>

  <attribute name="LoadBalancingFactory">
    org.acme.MyLoadBalancingFactory
  </attribute>

  <attribute name="PrefetchSize">1000</attribute>

  <attribute name="SlowConsumers">false</attribute>

  <attribute name="StrictTck">true</attribute>

  <attribute name="SendAcksAsync">false</attribute>

  <attribute name="DefaultTempQueueFullSize">50000</attribute>

  <attribute name="DefaultTempQueuePageSize">1000</attribute>

  <attribute name="DefaultTempQueueDownCacheSize">1000</attribute>

  <attribute name="DupsOKBatchSize">10000</attribute>
</mbean>
```

5.10.1. ConnectionFactory Managed Bean Attributes

ClientID

You can preconfigure a connection factory with a client ID. Any connection created via this connection factory will obtain this client ID.

JNDIBindings
: Lists available JNDI bindings for this connection factory.

PrefetchSize
: Specifies how many messages the window holds at once, for consumer flow control. The window size determines the number of messages a server can send to a consumer without blocking. Each consumer maintains a buffer of messages from which it consumes.

  Transmission Control Protocol (TCP) implements its own additional flow control. Message consumption can also be blocked if the TCP window size is smaller than the *`PrefetchSize`* parameter.

SlowConsumers
: Specifies whether the allowable buffer size for slow consumers is reduced. Reducing the buffer size for slow consumers results in minimized to increase the potential for messages to be consumed by faster consumers. It is not possible to totally disable buffering, however, setting the `SlowConsumers` attribute to **`true`** will reduce the buffer size. Setting this attribute to **`true`** is equivalent to setting `PrefetchSize` to **`1`** which is the lowest possible value available.

StrictTck
: Enables strict JMS behavior if the attribute is set to **`true`**. Strict JMS behavior is required by the Technology Compatibility Kit (TCK).

SendAcksAsync
: Specifies acknowledgments are sent asynchronously if the attribute is set to **`true`**. This can improve performance, particularly if **`auto_acknowledge`** mode is active.

DefaultTempQueueFullSize
: Optional attribute that specifies the paging parameters for temporary full size queue destinations, which are scoped to connections created with this connection factory. The default value is **`200000`**. For more information about these attributes, refer to *Section 5.9.3.1.2, “Destination paging parameters”*.

DefaultTempQueuePageSize
: Optional attribute that specifies the paging parameters for temporary page size destinations, which are scoped to connections created with this connection factory. The default value is **`2000`**. For more information about these attributes, refer to *Section 5.9.3.1.2, “Destination paging parameters”*.

DefaultTempQueueDownCacheSize
: Optional attribute that specifies the paging parameters for temporary down cache size destinations, which are scoped to connections created with this connection factory. The default value is **`2000`**. For more information about these attributes, refer to *Section 5.9.3.1.2, “Destination paging parameters”*.

DupsOKBatchSize
: Specifies the number of **`DUPS_OK_ACKNOWLEDGE`** acknowledgments that are buffered locally before they are sent. The default value is **`2000`**.

SupportsLoadBalancing
: Specifies whether client-side load balancing is enabled for the connection factory on clustered installations. If load balancing is enabled, any connection created by that connection factory will be load-balanced across the nodes of a cluster. A connection created on a particular node remains on that node. The default value is **`false`**.

SupportsFailover

Specifies whether client-side automatic failover is enabled for the connection factory on clustered installations. If automatic failover is enabled, JBoss Messaging will automatically and transparently failover to another node in the cluster when a connection problem is detected. The default value is **`false`**.

Note

When automatic failover is disabled, the user code is responsible for catching connection exceptions in synchronous JMS operations, and a JMS **`ExceptionListener`** must be installed to catch exceptions asynchronously. When an exception is caught, the client-side code must lookup a new connection factory via HAJNDI and recreate the connection.

DisableRemotingChecks

Specifies whether the connection factory checks that the corresponding JBoss Remoting Connector uses sensible values. JBoss Messaging is very sensitive to these values, and there is rarely any need to change them. To disable this sanity checking, set `DisableRemotingChecks` to **`false`**. The default value is **`true`**.

Warning

Do not disable the remoting checks; system instability.

LoadBalancingFactory

Specifies the client-side load balancing factory implementation used by the connection factory. The value must correspond to the name of a class that implements the interface **`org.jboss.jms.client.plugin.LoadBalancingFactory`**.

The default value is **`org.jboss.jms.client.plugin.RoundRobinLoadBalancingFactory`**, which load-balances connections across the cluster in a round-robin fashion.

Connector

Specifies the remoting connector used by the connection factory. Different connection factories can use different connectors, so you can deploy one connection factory that uses the HTTP transport to communicate with the server, and another that uses the bisocket transport to communicate.

EnableOrderingGroup

Specifies whether strict message ordering is enabled on the **`ConnectionFactory`**. If set to **`true`**, any messages sent from producers which are created from the enabled connection factory become ordering group messages. The default value for this parameter is **`false`**.

DefaultOrderingGroupName

Specifies the default name for the message ordering group. The specified name will take effect once the **`EnableOrderingGroup`** parameter is set to **`true`** . If this attribute is missing, the group name will be generated automatically.

5.11. Configuring the Remoting Connector

JBoss Messaging uses JBoss Remoting for all communication between the client and the server.

For more information about JBoss Remoting configuration and capabilities, refer to the *Remoting* chapter in the *Administration and Configuration Guide*.

The default configuration includes one remoting connector, which is used by the single default connection factory. Each connection factory can be configured to use a different connector.

The default connector is configured to use the remoting bisocket transport, a TCP socket-based transport that listens and accepts connections only on the server side. That is, connections are always initiated from the client side. This connector is ideal for typical firewall scenarios, where only inbound connections are allowed on the server, or where only outbound connections are allowed from the client.

The bisocket transport can be configured to use SSL when a higher level of security is required.

The other supported transport is the HTTP transport, which uses the Hypertext Transfer Protocol to communicate between client and server. The client periodically polls the server for messages to receive data. This transport is ideal when a firewall between server and client allows only incoming HTTP traffic on the server. Because of its polling behavior and the limitations of HTTP, this transport does not perform as well as the bisocket transport. It is not designed to handle high-load situations.

No other remoting transports are currently supported by JBoss Messaging.

Remoting configuration details can be seen in ***$JBOSS_HOME*/server/*$SERVER*/deploy/messaging/remoting-bisocket-service.xml**. The following code is an example of a bisocket remoting configuration:

Example 5.2. Bisocket Remoting Configuration

```
        <?xml version="1.0" encoding="UTF-8"?>

<!--
     Standard bisocket-based Remoting service deployment descriptor.

     $Id: remoting-bisocket-service.xml 3981 2008-03-28 18:00:41Z timfox $
 -->

<server>

   <!-- Standard bisocket connector - the bisocket transport only opens connection from
 client->server
        so can be used with firewalls where only outgoing connections are allowed.
        For examples of HTTP and SSL transports see docs/examples -->
   <mbean code="org.jboss.remoting.transport.Connector"
          name="jboss.messaging:service=Connector,transport=bisocket"
          display-name="Bisocket Transport Connector">
      <attribute name="Configuration">
         <config>
            <invoker transport="bisocket">

               <!-- There should be no reason to change these parameters - warning!
                    Changing them may stop JBoss Messaging working correctly -->

               <attribute name="marshaller"
 isParam="true">org.jboss.jms.wireformat.JMSWireFormat</attribute>
               <attribute name="unmarshaller"
 isParam="true">org.jboss.jms.wireformat.JMSWireFormat</attribute>
               <attribute name="dataType" isParam="true">jms</attribute>
```

```
               <attribute name="socket.check_connection" isParam="true">false</attribute>

               <attribute name="serverBindAddress">${jboss.bind.address}</attribute>
               <attribute name="serverBindPort">
${jboss.messaging.connector.bisocket.port:4457}</attribute>
               <attribute name="clientSocketClass"
 isParam="true">org.jboss.jms.client.remoting.ClientSocketWrapper</attribute>
               <attribute
 name="serverSocketClass">org.jboss.jms.server.remoting.ServerSocketWrapper</attribute>
               <attribute
 name="onewayThreadPool">org.jboss.jms.server.remoting.DirectThreadPool</attribute>

               <!-- the following parameters are useful when there is a firewall between
 client and server. Uncomment them if so.-->
               <!--
               <attribute name="numberOfCallRetries" isParam="true">1</attribute>
               <attribute name="pingFrequency" isParam="true">214748364</attribute>
               <attribute name="pingWindowFactor" isParam="true">10</attribute>
               <attribute name="generalizeSocketException" isParam="true">true</attribute>
               -->

               <!-- Now remoting supports socket write timeout configuration. Uncomment
 this if you need it. -->
               <!--
               <attribute name="writeTimeout" isParam="true">30000</attribute>
               -->

               <!-- End immutable parameters -->

               <attribute name="stopLeaseOnFailure" isParam="true">true</attribute>

               <!-- Periodicity of client pings. Server window by default is twice this
 figure -->
               <attribute name="clientLeasePeriod" isParam="true">10000</attribute>
               <attribute name="validatorPingPeriod" isParam="true">10000</attribute>
               <attribute name="validatorPingTimeout" isParam="true">5000</attribute>

               <attribute name="failureDisconnectTimeout" isParam="true">0</attribute>
               <attribute name="callbackErrorsAllowed">1</attribute>
               <attribute name="registerCallbackListener">false</attribute>
               <attribute name="useClientConnectionIdentity" isParam="true">true</
attribute>

            <attribute name="timeout" isParam="true">0</attribute>

               <!-- Max Number of connections in client pool. This should be significantly
 higher than
                    the max number of sessions/consumers you expect -->
               <attribute name="JBM_clientMaxPoolSize" isParam="true">200</attribute>

               <!-- The maximum time to wait before timing out on trying to write a
 message to socket for delivery -->
               <attribute name="callbackTimeout">10000</attribute>

               <!-- Use these parameters to specify values for binding and connecting
 control connections to
                    work with your firewall/NAT configuration
               <attribute name="secondaryBindPort">xyz</attribute>

               <attribute name="secondaryConnectPort">abc</attribute>
               -->

           </invoker>
           <handlers>
              <handler
 subsystem="JMS">org.jboss.jms.server.remoting.JMSServerInvocationHandler</handler>
           </handlers>
```

```
         </config>
      </attribute>
   </mbean>

</server>
```

There are restricted attributes that should not be changed unless you are absolutely confident you understand the impact of the changes. The following attributes are safe to change and configure to the requirements of your project:

clientLeasePeriod
: Clients periodically return *heartbeats* to the server to confirm that they are still active. If the server does not receive a heartbeat after a certain period of time, it will close down the connection and remove all resources that correspond to the client's session. The `clientLeasePeriod` determines the period of time between heartbeats, in milliseconds. The default value is **`10000`**.

  By default, the server closes a client if it does not receive a heartbeat within double the `clientLeasePeriod`. In reality, the period is automatically resized according to system load.

numberOfRetries
: The number of seconds JBoss Remoting blocks on the client pool while waiting for a connection to become available. If you have a very large number of sessions concurrently accessing the server from a client and cannot obtain connections from the pool, you may want to increase this value.

clientMaxPoolSize
: JBoss Remoting maintains a client-side pool of TCP connections on which to service requests. If you have a large number of sessions concurrently accessing the server from a client and cannot obtain connections from the pool, you may want to increase this value.

secondaryBindPort
: The bisocket transport uses control connections to pass control messages between server and client. This attribute defines the address to which the secondary **`ServerSocket`** is bound. To work behind a firewall, you may need to specify a particular value for your firewall configuration.

secondaryConnectPort
: The port that the client uses to connect. Specify this to let your client work with NAT routers.

maxPoolSize
: The number of threads used on the server side to service requests.

By default, JBoss Messaging binds to **`${jboss.bind.address}`**, which can be defined by running the **`./run.sh -c [yourconfig] -b [yourIP]`** command.

If necessary, you can change **`remoting-bisocket-service.xml`** to use a different communication port.

Warning

Do not change values in the connector configuration other than those listed above. Changing other values can cause JBoss Messaging to stop functioning correctly.

5.12. ServiceBindingManager

The **`SeviceBindingManager`** provides multiple application server instances running on the same IP using different port ranges, which is useful during development. There are other ways to do this, but the **`ServiceBindingManager`** removes much hassle.

5.13. Message Driven Beans

A message-driven bean is an enterprise bean that allows J2EE applications to process messages asynchronously. It acts as a JMS message listener, which is similar to an event listener except that it receives messages instead of events. The messages may be sent by any J2EE component--an application client, another enterprise bean, or a Web component--or by a JMS application or system that does not use J2EE technology. This definition is from *http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/EJBConcepts5.html* , and you can read more about message driven beans (MDB) from there.

You can specify MDBs in a deployment descriptor or using annotations.

Using a descriptor

```
           <enterprise-beans>
  <message-driven>
    <ejb-name>MDBExample</ejb-name>
    <destination-jndi-name>queue/@QUEUE_NAME@</destination-jndi-name>
  </message-driven>
</enterprise-beans>
```

Using an annotation

```
           @MessageDriven(mappedName="jms/Queue")
public class SimpleMessageBean implements MessageListener {
    @Resource
    private MessageDrivenContext mdc;
    ...
```

You configure MDBs using properties divided into those specified by the JCA specification, and those available as JBoss extensions.

Important

MDB properties listed in *Table 5.1, "MDB Properties Provided by the JCA Specification"* are not mandatory, unless explicitly called out in the Remarks column.

Table 5.1. MDB Properties Provided by the JCA Specification

Name	Type	Default value	Remarks
destination	java.lang.String	none	This property is Mandatory The JNDI name of the Queue or Topic.
destinationType	java.lang.String	none	The type of destination valid values are javax.jms.Queue or javax.jms.Topic
messageSelector	java.lang.String	none	The message selector of the subscription
acknowledgeMode	int	AUTO_ ACKNOWLEDGE	The type of acknowledgement when not using transacted jms - valid values AUTO_ ACKNOWLEDGE or DUPS_OK_ ACKNOWLEDGE
clientID	java.lang.String		The client id of the connection
subscriptionDurability	String	NonDurable	Whether topic subscriptions are durable. Valid values are Durable or NonDurable
subscriptionName	String	none	The subscription name of the topic subscription

Important

MDB properties listed in *Table 5.2, "MDB Properties Provided as JBoss Extensions"* are not mandatory, unless explicitly called out in the Remarks column.

Table 5.2. MDB Properties Provided as JBoss Extensions

Name	Type	Default value	Remarks
isTopic	boolean	false	Sets the destinationType
providerAdapterJNDI	java.lang.String	DefaultJMSProvider	The JNDI name of the JMS provider.
user	java.lang.String	none	The user ID used to connect to the JMS server
pass	java.lang.String	none	The password of the user
maxMessages	int	1	Read this number of messages before delivering messages to the MDB. Each message is delivered individually on the same thread in an attempt to avoid context excessive context switching
minSession	int	1	The minimum number of JMS sessions that are available to concurrently deliver messages to this mdb
maxSession	int	15	The maximum number of JMS sessions that are available to concurrently deliver messages to this mdb
reconnectInterval	long	10 seconds	The length of time in seconds between attempts to (re-)connect to the JMS provider
keepAlive	long	60 seconds	The length of time in milliseconds that sessions over the minimum are kept alive
sessionTransacted	boolean	true	Whether the sessions are transacted
useDLQ	boolean	true	Whether to use a Dead Letter Queue (DLQ) handler.
dLQJNDIName	java.lang.String	queue/DLQ	The JNDI name of the DLQ

Name	Type	Default value	Remarks
dLQHandler	java.lang.String	org.jboss.resource.adapter.jms.inflow.dlq.GenericDLQHandler	The org.jboss.resource.adapter.jms.inflow.DLQHandler implementation class name.
dLQUser	java.lang.String	none	The user id used to make the dlq connection to the JMS server
dLQPassword	java.lang.String	none	The password of the dLQUser
dLQClientID	java.lang.String	none	The client id of the DLQ connection
dLQMaxResent	int	5	The maximum number of times a message is redelivered before it is sent to the DLQ.
redeliverUnspecified	boolean	true	Whether to attempt to redeliver a message in an unspecified transaction context
transactionTimeout	int	Default is the timeout set for the resource manager	Time in seconds for the transaction timeout
DeliveryActive	boolean	true	Whether the MDB should make the subscription at initial deployment or wait for start() or stopDelivery() on the corresponding MBean. You can set this to false if you want to prevent messages from being delivered to the MDB (which is still starting) during server startup

Configuring default MDB properties

You can configure MDBs to have default properties using the `@org.jboss.ejb3.annotation.DefaultActivationSpecs` annotations.

Clustering Notes

To help locate clustering-related information, a summary of each consideration is provided in this part of the guide with links to the related components of JBoss Messaging.

6.1. Unique Server Peer ID

In most cases, JBoss Messaging works in a clustered environment with minimal configuration changes. One crucial change that must be made is that every node is assigned a unique server ID.

Every deployed node must have a unique ID, including nodes that form a LAN cluster and nodes linked by message bridges.

The *`ServerPeerID`* attribute is used to set this information. Refer to *Section 5.2, "ServerPeer attributes"* for further information.

6.2. Clustered Destinations

JBoss Messaging clusters Java Message Service (JMS) queues and topics transparently across the cluster. Messages sent to a distributed queue or topic on one node are consumable on other nodes. To make a particular destination clustered, the `clustered` attribute is used to set this functionality. Refer to *Section 5.5.1, "MessagingPostOffice Attributes"* for further information.

JBoss Messaging balances messages between nodes and caters for consumers of varying speeds so processing load can be efficiently distributed across the cluster.

To disable message redistribution between nodes while retaining other characteristics of clustered destinations, do not specify the `ClusterPullConnectionFactoryName` attribute on the Server Peer. Refer to *Section 5.2, "ServerPeer attributes"* for full details about this attribute.

6.3. Clustered Durable Subscriptions

JBoss Messaging durable subscriptions can be clustered in a way that allows multiple subscribers on multiple nodes to consume from one durable subscription. A durable subscription is clustered automatically, providing its topic is clustered.

For more information about configuring clustered topics and queues, refer to the *`Clustered`* attribute in *Section 5.5.1, "MessagingPostOffice Attributes"*

6.4. Clustered Temporary Destinations

JBoss Messaging supports clustering of temporary topics and queues. All temporary topics and queues will be clustered if the Post Office is clustered.

For more information about configuring clustered topics and queues, refer to the *`Clustered`* attribute in *Section 5.5.1, "MessagingPostOffice Attributes"*.

6.5. Non-clustered Servers

Set the *`PostOffice`* clustered attribute to **`false`** if you do not want all nodes to participate in a cluster, or if you do not want the server to be clustered.

For more information about configuring non-clustered server, refer to the various attributes in *Section 5.5.1, "MessagingPostOffice Attributes"*.

6.6. Message Ordering in the Cluster

To ensure messages are consumed in the same order they were produced, set strict JMS ordering by setting the `DefaultPreserveOrdering` Server Peer attribute to **`true`**. While set to **`true`**, messages cannot be distributed as freely around the cluster. The default value is **`false`**.

6.7. Idempotent Operations

A message is guaranteed to be persisted when the message sent to a persistent destination returns with no exception.

An exception does not guarantee the message was *not* persisted, because failure may have occurred between the message being persisted and a response being returned to the caller.

Applications must therefore be coded so that operations are *idempotent* — that is, operations can be repeated without causing the system to become inconsistent.

You can implement this behaviour on the application level by checking for duplicate messages and discarding them if the original message has been sent successfully. This *duplicate checking* is a powerful technique that can remove the need for XA transactions.

JBoss Messaging is configured by default to perform duplicate checking in a clustered environment.

Persistence considerations are located in *Section 5.3, "ServerPeer methods"*, *Section 5.4, "Changing the Database"*, *Section 5.7, "Configuring the Persistence Manager"*, and *Section 8.1, "Message Bridge Overview"*.

6.8. Clustered Connection Factories

When `supportsLoadBalancing` is set to **`true`** in the connection factory, consecutive attempts to create connections will round-robin between available servers. The first node is chosen randomly.

When `supportsFailover` is set to **`true`**, failover will occur transparently and automatically whenever any connection error is detected.

For more information about configuring connection factories, refer to *Section 5.10.1, "ConnectionFactory Managed Bean Attributes"*.

JBoss Messaging XA Recovery Configuration

This section describes how to configure JBoss Transactions to handle XA recovery for JBoss Messaging resources in JBoss Enterprise Application Platform.

The JBoss Transactions Recovery Manager can be configured to continually poll for and recover JBoss Messaging XA resources. This provides a high level of transaction durability.

To enable JBoss Transactions Recovery Manager, add a line to **$JBOSS_HOME/server/$PROFILE/conf/jbossts-properties.xml**. The following code snippet includes the line required:

```
<properties depends="arjuna" name="jta">
  <!--
  Support subtransactions in the JTA layer?
  Default is NO.
  -->
  <property name="com.arjuna.ats.jta.supportSubtransactions" value="NO"/>
  <property name="com.arjuna.ats.jta.jtaTMImplementation"
      value="com.arjuna.ats.internal.jta.transaction.arjunacore.TransactionManagerImple"/>
  <property name="com.arjuna.ats.jta.jtaUTImplementation"
      value="com.arjuna.ats.internal.jta.transaction.arjunacore.UserTransactionImple"/>
  <!--
      *** Add this line to enable recovery for JMS resources using DefaultJMSProvider ***
  -->
  <property name="com.arjuna.ats.jta.recovery.XAResourceRecovery.JBMESSAGING1"
      value="org.jboss.jms.server.recovery.MessagingXAResourceRecovery;java:/
DefaultJMSProvider"/>

</properties>
```

Here, the Recovery Manager attempts to recover JMS resources via the JMS Provider Loader, **DefaultJMSProvider**.

DefaultJMSProvider ships with JBoss Enterprise Application Platform. It is defined in **$JBOSS_HOME/server/$PROFILE/conf/jms-ds.xml** (or, in a clustered environment, **hajndi-jms-ds.xml**). To perform recovery with a different JMS provider loader (for example, one that corresponds with a remote JMS Provider), add another line to the properties file and specify your remote provider instead of **DefaultJMSProvider**. Your provider's name should be listed in its managed bean configuration file.

Each provider requires a unique name, for example, **com.arjuna.ats.jta.recovery.XAResourceRecovery.JBMESSAGING1**, **com.arjuna.ats.jta.recovery.XAResourceRecovery.JBMESSAGING2**, etc.

Recovery should work with any JMS provider that implements recoverable XAResources (that is, it properly implements **XAResource.recover()**).

For the Recovery Manager to recover from any node of the cluster, you must add a line in **hajndi-jms-ds.xml** for every node of the cluster.

JBoss Messaging Message Bridge Configuration

8.1. Message Bridge Overview

JBoss Messaging includes a fully functional message bridge.

The bridge consumes messages from a source queue or topic and sends them to a target queue or topic, typically on a different server. The source and target servers do not need to be in the same cluster, so bridging is a reliable method of sending messages from one cluster to another (across a WAN, for example) and where the connection may be unreliable.

A bridge is deployed as a managed bean within any JBoss Enterprise Application Platform instance. To deploy, add the managed bean descriptor into the **deploy** directory of an Enterprise Application Platform configuration that contains JBoss Messaging.

The example in **$JBOSS_HOME/docs/examples/jboss-messaging-examples/bridge/** demonstrates a simple bridge deployed in JBoss Enterprise Application Platform and moving messages from the source to the target destination.

The bridge can also be used to retrieve messages from other non-JBoss Messaging JMS servers as long as they are JMS 1.1 compliant.

The bridge has built-in failure recovery; if the source or target server connection is lost, the bridge will attempt to reconnect to the source or target until it comes back online, at which point normal operation will resume.

The bridge can be configured to consume messages matching a particular JMS selector.

It can be configured to consume from a queue or a topic. When the bridge consumes from a topic, it can be configured to consume with a non-durable or a durable subscription.

The bridge can be configured to handle messages with one of three *quality of service* (QoS) levels:

Bridge QoS Levels

QOS_AT_MOST_ONCE
: This mode specifies that messages will arrive at the destination once at the most. Messages are consumed from the source and acknowledged before they are sent to the destination. Messages can be lost if failure occurs between the message leaving the source and arriving at the destination. Messages will therefore be delivered once at most.

  This mode is available for both persistent and non-persistent messages.

QOS_DUPLICATES_OK
: This mode specifies that messages are consumed from the source and acknowledged after they have been successfully sent to the destination. If failure occurs between a message arriving, and being acknowledged by the destination, that message is sent a second time when the system recovers.

  This mode is available for both persistent and non-persistent messages.

QOS_ONCE_AND_ONLY_ONCE
: This mode specifies that messages will arrive exactly once. When the message source and destination are on the same JBoss Messaging server instance, the message can be sent and received in the same local transaction.

If the source and destination are on different servers, you can implement message high durability by using a JTA transaction controlled by JBoss Transactions JTA implementation. If JTA is required, both connection factories must be **`XAConnectionFactory`** implementations.

This mode is only available for persistent messages.

This mode requires logging on both the transaction manager and the resource side to support recovery. If you require this level of QOS, you must enable XA Recovery with JBoss Transactions.

Alternative Methods

You may be able to apply *once and only once* semantics to a specific application, without setting **`QOS_ONCE_AND_ONLY_ONCE`**. Set **`QOS_DUPLICATES_OK`** mode, and then check for and discard duplicate messages at the destination.

You can implement **`QOS_ONCE_AND_ONLY_ONCE`** behaviour at the application level by maintaining a cache of received message IDs on disk and comparing received messages to this cache. Because the cache would only be valid for a certain period of time, this approach is not infallible, but can be a useful alternative depending on your application.

8.2. Bridge deployment

You can deploy a message bridge by adding a managed bean descriptor into the **`deploy`** directory of the JBoss Enterprise Application Platform installation that contains JBoss Messaging.

8.3. Bridge Configuration

The following code is an example configuration of the message bridge, showing all attributes. Some attributes have been commented out for this configuration, since not all attributes should be specified at once.

Example 8.1. Message Bridge Configuration

```
<mbean code="org.jboss.jms.server.bridge.BridgeService"
  name="jboss.messaging:service=Bridge,name=TestBridge"
  xmbean-dd="xmdesc/Bridge-xmbean.xml">

  <!-- The JMS provider loader that is used to lookup the source
       destination
  -->
  <depends optional-attribute-name="SourceProviderLoader">
  jboss.messaging:service=JMSProviderLoader,name=JMSProvider</depends>

  <!-- The JMS provider loader that is used to lookup the target
       destination
  -->
  <depends optional-attribute-name="TargetProviderLoader">
  jboss.messaging:service=JMSProviderLoader,name=JMSProvider</depends>

  <!-- The JNDI lookup for the source destination -->
  <attribute name="SourceDestinationLookup">/queue/A</attribute>

  <!-- The JNDI lookup for the target destination -->
  <attribute name="TargetDestinationLookup">/queue/B</attribute>

  <!-- The username to use for the source connection
```

```
<attribute name="SourceUsername">bob</attribute>
-->

<!-- The password to use for the source connection
<attribute name="SourcePassword">BobSecur3</attribute>
-->

<!-- The username to use for the target connection
<attribute name="TargetUsername">mary</attribute>
-->

<!-- The password to use for the target connection
<attribute name="TargetPassword">MaryS3cur3</attribute>
-->

<!-- Optional: The Quality Of Service mode to use, one of:
  QOS_AT_MOST_ONCE = 0;
  QOS_DUPLICATES_OK = 1;
  QOS_ONCE_AND_ONLY_ONCE = 2;
-->
<attribute name="QualityOfServiceMode">0</attribute>

<!-- JMS selector to use for consuming messages from the source
<attribute name="Selector">specify jms selector here</attribute>
-->

<!-- The maximum number of messages to consume from the source
     before sending to the target
-->
<attribute name="MaxBatchSize">5</attribute>

<!-- The maximum time to wait (in ms) before sending a batch to the
     target even if MaxBatchSize is not exceeded. -1 means wait forever
-->
<attribute name="MaxBatchTime">-1</attribute>

<!-- If consuming from a durable subscription this is the subscription
     name
     <attribute name="SubName">mysub</attribute>
-->

<!-- If consuming from a durable subscription this is the client ID to
     use
     <attribute name="ClientID">myClientID</attribute>
-->

<!-- The number of ms to wait between connection retrues in the event
     connections to source or target fail
-->
<attribute name="FailureRetryInterval">5000</attribute>

<!-- The maximum number of connection retries to make in case of failure,
     before giving up -1 means try forever
-->
<attribute name="MaxRetries">-1</attribute>

<!-- If true then the message ID of the message before bridging will be
     added as a header to the message so it is available to the
     receiver. Can then be sent as correlation ID to correlate in a
     distributed request-response
-->
<attribute name="AddMessageIDInHeader">false</attribute>

  </mbean>
```

Message Bridge Configuration Attributes

SourceProviderLoader, TargetProvider Loader

The **`JMSProviderLoader`** managed bean is used by the bridge to look up the source connection factory and source destination. By default, JBoss Enterprise Application Platform ships with one **`JMSProviderLoader`**, which is deployed in the **`$JBOSS_HOME/server/$PROFILE/deploy/messaging/jms-ds.xml`** file, and serves as the default local **`JMSProviderLoader`**. For a clustered configuration, **`hajndi-jms-ds.xml`** performs the same role.

If your source or target destination is on a different server, or corresponds to a non-JBoss JMS Provider, you can deploy another **`JMSProviderLoader`** managed bean instance that the bridge can use to contact the destination on the remote JMS Provider.

To use **`QOS_ONCE_AND_ONLY_ONCE`** delivery with a remote non-JBoss Messaging source or target, the remote JMS Provider must provide a fully-functional JMS XA resource implementation that works remotely from the server.

SourceDestinationLookup

The full JNDI lookup for the source destination, via the **`SourceProviderLoader`**, such as **`/queue/mySourceQueue`**.

TargetDestinationLookup

The full JNDI lookup for the target destination, via the **`TargetProviderLocator`**, such as **`/topic/myTargetTopic`**.

SourceUsername

An optional attribute that specifies the username used when creating the source connection.

SourcePassword

An optional attribute that specifies the password used when creating the source connection.

TargetUsername

An optional attribute that specifies the username used when creating the target connection.

TargetPassword

An optional attribute that specifies the password used when creating the target connection.

QualityOfServiceMode

An integer representing the desired *quality of service* mode. The possible values are:

- **`0`** to represent **`QOS_AT_MOST_ONCE`**
- **`1`** to represent **`QOS_DUPLICATES_OK`**
- **`2`** to represent **`QOS_ONCE_AND_ONLY_ONCE`**

See *Section 8.1, "Message Bridge Overview"* for a complete explanation of these modes.

Selector

An optional attribute that lets you provide a JMS selector expression when consuming messages from a source destination. Only messages that match the selector expression are bridged from the source to the target destination. The selector expression must follow the JMS selector syntax, specified here: *http://java.sun.com/j2ee/1.4/docs/api/javax/jms/Message.html*.

For optimal performance, apply source topic subscription selectors to source queue consumers.

MaxBatchSize
: Specifies the maximum number of messages to consume from the source destination before sending a message batch to the target destination. Its value must be greater than or equal to **1**.

MaxBatchTime
: Specifies the longest period (in milliseconds) to wait before sending a message batch to the target, even if the `MaxBatchSize` has not been reached. Its value must be either **-1** (wait forever) or greater than or equal to **1** to specify a time.

SubName
: Represents the name of the durable subscription that will consume from the source destination topic.

ClientID
: Represents the JMS client ID to use when creating or looking up the durable subscription that will consume from the source destination topic.

FailureRetryInterval
: The period of time (in milliseconds) to wait between attempting to recreate the connection to the source or target server after failure is detected.

MaxRetries
: The number of times to attempt to recreate the connection to the source or target server after failure is detected. The bridge will then stop attempting to recreate the connection. A value of **-1** means that the bridge will continue to attempt to reconnect forever.

AddMessageIDInHeader
: When **true**, the original message ID is added to the **JBossMessage.JBOSS_MESSAGING_BRIDGE_MESSAGE_ID_LIST** header of the message being sent to the destination. If the message is bridged multiple times, each message ID is added to the header. This enables a distributed request-response pattern.

Enabling JBoss Messaging Ordering Group

This section describes how to use the JBoss Messaging ordering group feature to achieve strict message ordering.

Message ordering groups is the JBoss Messaging implementation of strict message ordering. When the ordering group feature is enabled, message priorities no longer have an influence on the order that the messages are delivered. Messages of a particular ordering group are delivered in the exact order that they arrive at the target queue (FIFO).

The next message of an ordering group is delivered only after the delivery of the previous messages has been accomplished. To signalize a successful message delivery, the acknowledge mechanism is used (refer to *Section 9.1, "Acknowledge Mechanism"*).

Transactional Receipt

In the case of the transactional receipt of messages, the next message is not delivered until the transaction has been committed which includes the acknowledgment of the receipt of the current message. If the transaction is rolled back, the message is canceled, sent back to the JMS server, and made available for the next delivery.

9.1. Acknowledge Mechanism

The messages that form a part of an ordering group are delivered one at a time. The next message is delivered only after the delivery of a previous message was completed. Message delivery completion is signaled by various means, depending on the acknowledge mode settings:

- In the **`CLIENT_ACKNOWLEDGE`** mode, the completion state is signalized by the `Message.acknowledge()` method.
- In the **`AUTO_ACKNOWLEDGE`** and **`DUPS_OK_ACKNOWLEDGE`** modes, the completion is signalized by either of the following:
  - a successful return from one of the **`MessageConsumer`**`.receive()` methods, or
  - a successful return from the `onMessage()` call of the **`MessageListener()`**.

Note

If the message consumer is closed, the message being processed at the time of its closure is deemed as completed and that regardless of whether **`*_ACKNOWLEDGE`** was called prior to the closure of the consumer.

9.2. How to Enable Message Ordering Group

You can enable an ordering group either on a connection factory or on a producer:

- If you define the ordering group on a connection factory, all producers on the connection factory use the same ordering group (refer to *Section 9.2.1, "Enabling Ordering Group on Producer"*).

- If you define the ordering group on a producer (note that the producer is defined on a connection factory), the producer uses the defined ordering group. If the connection factory of the producer defines an ordering group as well, the producer ordering group overrides the setting and uses its own ordering group (refer to *Section 9.2.2, "Enabling Ordering Group on Connection Factory"*).

9.2.1. Enabling Ordering Group on Producer

To enable the ordering group feature on producer, do the following:

1. Add a **`JBossMessageProducer`** to your session:

```
JBossMessageProducer producer=(JBossMessageProducer)session.createProducer(queue);
```

2. Add the enableOrderingGroup method to set an ordering group:

```
producer.enableOrderingGroup(String ogrpName) throws JMSException
```

   The method creates an ordering group; when the method is called, JBossMessageProducer sends messages on behalf of the ordering group. If you provide **`null`** as the method parameter, the ordering group is generated automatically. A new call to this method overrides the previous call.

3. Optionally, you can add the `disableOrderingGroup()` method to disable the ordering group:

```
public void disableOrderingGroup() throws JMSException
```

   When the method is called, JBossMessageProducer stops sending ordering group messages and resumes its default behavior.

An example is available in **`$EAPHOME/doc/examples/jboss-messaging-examples/ordering-group/`** in the class **`OrderingGroupExample.java`** (**`src/org/jboss/example/jms/ordering/OrderingGroupExample.java`**). Note that the example is delivered as part of the **`jboss-eap-docs`** archive.

9.2.2. Enabling Ordering Group on Connection Factory

To enable the ordering group feature on a connection factory, add the following attributes to the factory service configuration file:

EnableOrderingGroup
: enables the ordering group feature if set to **`true`** (the default value is **`false`**).

DefaultOrderingGroupName
: sets the default name of the message ordering group. The group name is generated automatically if the attribute is not defined.

Note

Once configured to enable the ordering group feature on a connection factory, all messages that are sent from any producers created from the connection factory become ordering group messages.

The following factory service configuration file sample demonstrates how to enable the ordering group feature:

```
<mbean code="org.jboss.jms.server.connectionfactory.ConnectionFactory";
  name="jboss.messaging.connectionfactory:service=ConnectionFactory";
  xmbean-dd="xmdesc/ConnectionFactory-xmbean.xml">
  <depends optional-attribute-name="ServerPeer">
    jboss.messaging:service=ServerPeer
  </depends>
  <depends optional-attribute-name="Connector">
    jboss.messaging:service=Connector,transport=bisocket
  </depends>
  <depends>
    jboss.messaging:service=PostOffice
  </depends>

  <attribute name="JNDIBindings">
    <bindings>
    <binding>/MyConnectionFactory</binding>
    <binding>/XAConnectionFactory</binding>
    <binding>java:/MyConnectionFactory</binding>
    <binding>java:/XAConnectionFactory</binding>
    </bindings>
  </attribute>

  <!-- The two OrderingGroup properties -->
  <attribute name="EnableOrderingGroup">true</attribute>
  <attribute name="DefaultOrderingGroupName">MyOrderingGroup</attribute>
</mbean>
```

The advantage of enabling the ordering group feature by making configuration changes is the ease with which message ordering functionality can be achieved without the need for code changes.

9.3. Notes and Limitations

The following points should be noted in regard to ordering group functionality:

- Queues must be used with the ordering group feature. The feature will not work with topics.
- The ordering group feature should not be used in conjunction with message selectors and scheduled delivery.
- A message is considered completed, and the next message will be available for delivery, if the original message is dead or has expired. A dead message is moved to the **`DLQ`** whereas an expired message is moved to the **`ExpiryQueue`**.
- When using a **`ConnectionConsumer`**, the ordering of the messages will be observed. However, the **`ConnectionConsumer`** does not control which session will receive the next message.

- The ordering group feature does not work with clustered queues. However, if the user deploys the clustered queue as HASingleton, which means at a time only one queue of a clustered queue is active, ordering groups work as expected.

Appendix A. Revision History

Revision 5.1.2-100 **Thu 8 December 2011** **Russell Dickenson** ***rdickens at redhat dot com***

Incorporated changes for JBoss Enterprise Application Platform 5.1.2 GA. For information about documentation changes to this guide, refer to *Release Notes 5.1.2*.